VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MATEMATIKY
INSTITUTE OF MATHEMATICS

POČÍTAČOVÉ MODELOVÁNÍ 3D OBJEKTŮ POMOCÍ POVRCHOVÉ SÍTĚ

COMPUTER MODELING OF 3D OBJECTS USING SURFACE NETWORK

BAKALÁŘSKÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Hana Gurecká
AUTHOR

VEDOUCÍ PRÁCE — Ing. et Ing. Stanislav Lang
SUPERVISOR

BRNO 2018

ABSTRAKT

Práce je rozdělena do rešeršní a praktické části. Cílem rešerše je seznámení se s modelováním objektů pomocí povrchových sítí. Jsou rozebrány základní vlastnosti a typy struktur povrchových sítí. Značný prostor je věnován kapitole o algoritmech určených k redukci počtu trojúhelníků na povrchové síti. V rámci praktické části jsou implementovány algoritmy shlukování bodů a kolapsu hrany. Závěrem práce jsou diskutovány výsledky.

ABSTRACT

The thesis is divided into theoretical and practical part. The goal of the thesis is introduction to object modeling using surface meshes. The basics features and mesh data structures are discussed. Considerable space is devoted to the chapter about mesh decimation algorithms. In practical part of the thesis vertex clustering and edge collapse algorithms are implemented. In the end acquired results are disscussed.

KLÍČOVÁ SLOVA

Trojúhelníkové povrchové sítě, redukce povrchových sítí, bodové shluky, kolaps hrany.

KEYWORDS

Triangle meshes, srface mesh decimation, vertex clustering, edge collapse.

BIBLIOGRAFICKÁ CITACE

GURECKÁ, Hana. *Počítačové modelování 3D objektů pomocí povrchové sítě,* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav matematiky.

PODĚKOVÁNÍ

Děkuji svému vedoucímu Ing. et Ing. Stanislavu Langovi za odborné vedení a konzultace při sepisování práce. Dále děkuji konzultantovi z firmy B&R Ing. Václavovi Velebovi za cenné rady v oblasti programování.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracovala jsem ji samostatně pod vedením Ing. et Ing. Stanislava Langa a s použitím literatury uvedené v seznamu literatury.

V Brně dne 25.5.2018 ……

Hana Gurecká

OBSAH

1 ÚVOD

Práce bude rozdělena do dvou částí. První část bude rešeršní, zabývající se obecně reprezentaci objektů, speciálně povrchovými sítěmi a pojmy s danými tématy souvisejícími. Stěžejní část rešerše se bude zabývat algoritmy pro redukci počtu trojúhelníků na povrchové síti se zmínkou o dostupných softwarových knihovnách podporujících zjednodušování povrchových sítí.

V první kapitole bude nastíněn úvod do počítačové grafiky, možnosti grafické reprezentace objektů v počítači (body, povrchové sítě, povrchové křivky, objemové reprezentace těles, konstruktivní objemová geometrie, binární stromy) a jejich využití.

Druhá kapitola bude věnována již výhradně povrchovým sítím, kdy si přiblížíme výhody a nevýhody různých druhů povrchových sítí (trojúhelníkové, čtyřúhelníkové, mnohoúhelníkové). V obecné rovině si také představíme pojmy související s povrchovými sítěmi. Zběžně se podíváme na vlastnosti povrchových sítí (pevnost, propojenost, konvexnost, rovinnost, jednoduchost) a rozlišíme ty, které považujeme za nutný předpoklad povrchové sítě obecně a které budeme my předpokládat v rámci dalších částí práce. Dále si představíme pojmy jako geometrie a topologie sítě a rozdíl mezi nimi. Nebudou opomenuty základní podstruktury, které můžeme na sítích najít, ani bližší vysvětlení pojmu varietní sítě. V poslední části druhé kapitoly si představíme základní datové struktury pro reprezentaci povrchových sítí a práci s nimi. Zmíníme struktury založené na bodech, hranách i ploškách a osvětlíme si jejich vzájemné výhody a nevýhody.

Třetí kapitola bude zaměřena již výhradně na trojúhelníkové povrchové sítě a algoritmy určené k jejich redukci (sjednocení koplanárních ploch, shlukování bodů, algoritmy založená na cyklickém mazání bodů, hran nebo trojúhelníků, tvarová aproximace). Mimo jednotlivých algoritmů si představíme i chybové metriky, které se pro konkrétní algoritmus používají (Hausdorffova vzdálenost, vzdálenost od průměrové roviny, kvadratická chyba, funkce $\mathcal{L}^2$). Dále si jen velmi stručně shrneme knihovny, ve kterých najdeme některou funkci ke zjednodušování povrchových sítí.

Bude následovat praktická část práce, kde budou popsané implementace vybraných algoritmů. Poté budou srovnány jejich výsledky a bude diskutována jejich použitelnost a efektivita.

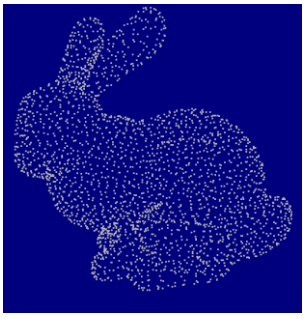
2 ÚVOD DO 3D GRAFIKY

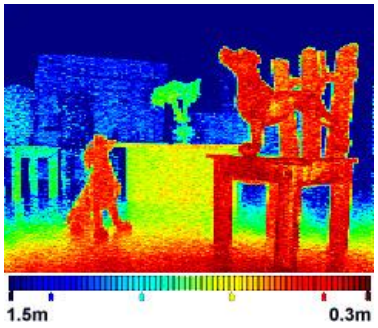
Trojrozměrné těleso je možné v počítačové grafice reprezentovat řadou způsobů. Tyto způsoby jsou různě výpočetně náročné a mají různá uplatnění. Pojďme si tedy představit základní možnosti reprezentace těles.

2.1 Surová data (Raw data)

Jedná se o reprezentaci těles získanou různými metodami snímání z reálných těles. Tato data podle potřeby mohou být sama vizualizačním výstupem, nebo jsou dále zpracována do jiné podoby.

- **Bodové mraky (Point clouds)**: Bodové shluky jsou velké netříděné množiny bodů v trojrozměrném prostoru, které velmi přesně reprezentují povrch skutečného objektu (Obr. 1). Jedná se o surová data například z 3D skenerů, která lze dále upravovat například do polygonálních mřížek. Pokud je shluk dostatečně hustý, případně body dostatečně velké je lidský mozek schopný shluky vnímat jako ucelený objekt. Výhodou bodové reprezentace je, že body mezi sebou nemají žádné vztahy (hrany, plochy, křivky,...), z čehož vyplývá malá výpočetní náročnost úprav, filtrace či zobrazování dat a to i přes velké množství bodů[1]. Bodové shluky nachází uplatnění například v oblasti kontroly kvality, metrologii atd.
- **Hloubkové mapy (Range images)**: V případě hloubkových map jsou data reprezentována pomocívzdáleností od referenčního bodu.Tyto vzdálenosti jsou získávány pomocí distančních senzorů. Ve výsledném obrazu je pak každému pixelu na základě této vzdálenosti přiřazena barva (Obr. 2). Hloubkových map se často využívá v počítačovém vidění, k detekci hran, nebo konstrukci 3D map.





Obr. 1: Bodový shluk[2]
Obr. 2: Hloubková mapa[3]

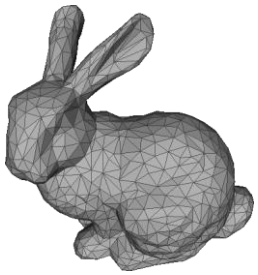
[1] In: *3Deling* [online]. [cit. 2018-05-25]. Dostupné z: http://www.3deling.com/whta-is-a-point-cloud/
[2] Dostupné z: https://www.codeproject.com/Articles/839389/Fast-Point-Cloud-Viewer-with-Csharp-and-OpenGL
[3] Dostupné z: http://www.idl.rie.shizuoka.ac.jp/study/project/tof/index_e.html

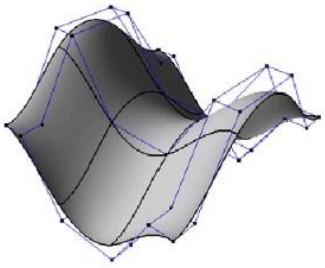
- **Mnohoúhelníkový mix (Polygon soup)**: Je speciálním případem povrchové sítě. Představuje neuspořádaný soubor mnohoúhleníků, které nemají definované žádné vzájemné vztahy. Výhodou mnohoúhleníkových mixů je menší náročnost na úložný prostor a významná úspora při výpočtech[4]. Dnes jsou nejčastěji využívány při procesech osvětlování a vykreslování například 3D grafickými programy jako Maya, Houdini nebo Blender.

2.2 Reprezentace povrchů objektů

Jde o reprezentaci tělesa, kdy nám stačí vidět pouze „slupku" zobrazovaného objektu. Jedná se tedy o jakýsi vnější obal, kdy vnitřek modelovaného tělesa je dutý.

- **Povrchová síť (Polygonal mesh)**: Je síť tvořenanejčastěji trojúhelníky (ale existují i obdoby v podobě jiných mnohoúhleníků) s určitou strukturou (topologií) (tím se liší od mnohoúhelníkových mixů) (Obr. 3). Mnohoúhelníkovou mřížkou se budeme podrobněji zabývat v kapitole 3. Výhody povrchové sítě je možnost modelovat téměř jakýkoliv objekt, snadná reprezentace, jednoduché transformace a vykreslování na obrazovku. Nevýhodami je, že planárními mnohoúhelníky je možné zakřivené části objektů pouze aproximovat a dále obtížnost simulace některých typů objektů (např. vlasy, tekutiny)[5].
- **NURBS (Non-uniform rational b-spline)**: Je reprezentace určená pro vykreslování hladkých ploch. NURBS povrchy jsou dvouparametrickými funkcemi vzniklými zobecněním B-splinů a Beziérových křivek. Povrch objektu je určen kontrolními body (Obr. 4).





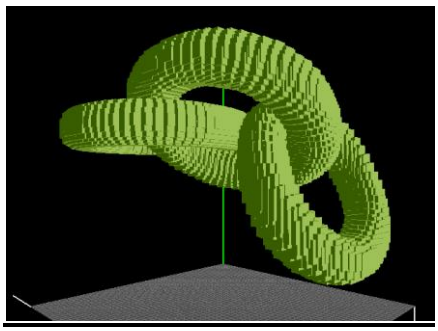
Obr. 3: Trojúhleníková povrchová síť[6]
Obr. 4: NURBS reprezentace[7]

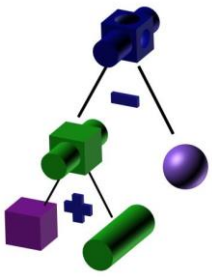
[4] PolySoup surface node. In: *Sidefix* [online]. [cit. 2018-05-25]. Dostupné z: http://www.sidefx.com/docs/houdini12.5/nodes/sop/polysoup
[5] *3D Object Representations* [online]. In: . 2012 [cit. 2018-05-25]. Dostupné z: http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/meshes/polygon_meshes.html
[6] Dostupné z: http://www.lix.polytechnique.fr/~maks/Verona_MPAM/TD/TD2/images/snapshot00.png
[7] Dostupné z: http://softimage.wiki.softimage.com/xsidocs/surfs_About_Surfaces.htm

2.3 Objemová reprezentace

V případech, kdy nestačí znát pouze povrchy těles, ale i jejich vnitřní objem, je volena objemová reprezentace těles.

- **Voxely**: Voxely jsou obdobou pixelů v třídimenzionálním prostoru. Je dána mřížka (buď s uniformním, nebo neuniformním rozdělením) a u každého bodu této mřížky máme informaci o tom, zda náleží, či nenáleží zobrazovanému tělesu (Obr.5). Výhodou reprezentace voxely je snadná detekce kolizí, není nutné znát přesný povrch objektu a operace jako výřez, sjednocení, rozdíl atd. jsou taktéž velmi snadné. Nevýhodou je velký objem dat při velkých objektech a jemném uniformním mřížkování. Voxelů je využíváno například v počítačových hrách nebo ve zdravotnictví (Voxel Based Morphometry -VBM), CT, MRI.
- **Konstruktivní objemová geometrie (CSG: constructive solid Geometry)**: CSG je metoda, kdy z tzv. primitivů, tj. jednoduchých objektů sestavujeme komplexnější objekty. Standardními primitivy jsou například krychle/kvádr, válec, kužel, koule torus atd. Sestavování složitějších objektů pak odpoví regularizovaným booleovským operacím: sjednocení, průnik, rozdíl. Jednotlivé primitivy a operace nad nimi jsou uloženy v grafové struktuře (Obr. 6). Výhodami CSG jsou velmi snadná konstrukce těles a díky úsporné struktuře i nízká náročnost na úložný prostor. Nevýhodou je možnost použití pouze booleovských operací, což má za následek značné omezení modelovatelných objektů[8]. CSG reprezentace se většinově využívá v inženýrských CAD (Computer Aided Design) softwarech.



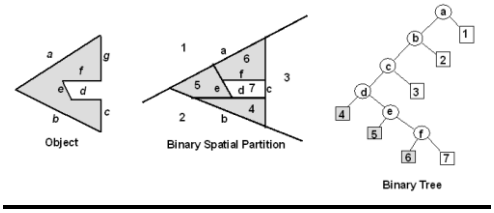


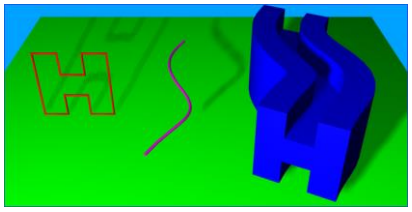
Obr. 5: Vyobrazení pomocí voxelů
Obr. 6: Princip konstruktivní objemové geometrie[9]

- **Binární prostor rozdělující stromy (SPP trees: Binary Space Partitioning trees)**: Jedná se o grafovou strukturu reprezentující mnohostěny pomocí rekurzivního dělení prostoru rovinami na konvexní podprostory (Obr.7). BSP stromů se hojně využívá například v počítačových hrách, kde je potřeba urychlení výpočtů a vykreslování v reálném čase.
- **Tažení (Sweep)**: Reprezentace objektu pomocí křivky nebo dvojrozměrného tvaru a křivky představující trajektorii. Objekt vznikne tak, že křivka nebo útvar je tažena

[8] Solid modelling cg. In: *Slideshare* [online]. 2016 [cit. 2018-05-25]. Dostupné z: https://www.slideshare.net/Nareek/solid-modelling-cg
[9] Dostupné z: http://groups.csail.mit.edu/graphics/classes/6.837/F98/talecture/

po trajektoriikřivky trajektorie (Obr. 8). Různorodost takto reprezentovaných objektů je zřejmě značně limitovaná, proto se metoda tažení používá spíše jako doplněk v CAD softwarech.





Obr. 7: Tvorba BSP stromu[10]
Obr. 8: Tažení

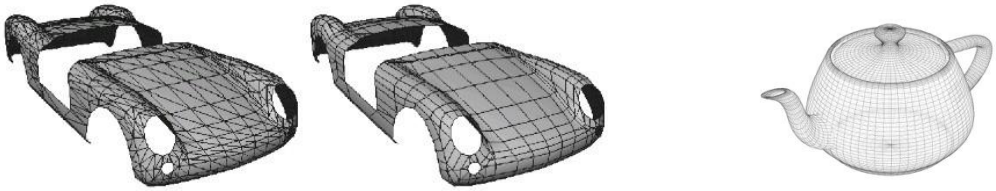
[10] Dostupné z: http://www.connellybarnes.com/work/class/2015/intro_gfx/lectures/17-3DObjectRepresentation.pdf

3 MODELOVÁNÍ 3D OBJEKTŮ POMOCÍ POVRCHOVÉ SÍTĚ

V minulé kapitole jsme si obecně představili možnosti trojrozměrné reprezentace objektů. Nyní se podstatně detailněji zaměříme na reprezentaci pomocí mnohoúhleníkové resp. trojúhelníkové povrchové sítě.

Síť polygonů je povrch zkonstruovaný z množiny bodů. Tyto body jsou propojeny hranami, které tvoří síť mnohoúhelníků generující povrch objektu. Nejčastěji bývají mnohoúhelníkové sítě zobrazovány pomocí černých čar s šedou výplní (Obr.9). Pokud by bylo zobrazení bez šedé výplně, jednalo by se o tzv. wireframe model (Obr.10). Jak již bylo vše zmíněno mřížka může být tvořena ve své podstatě libovolnými mnohoúhelníky, my se však dále zaměříme na trojúhelníky z několika prostých důvodů:

- Trojúhelníky jsou zdaleka nejpoužívanějšími mnohoúhelníky při tvorbě povrchových sítí.
- Pro povrchové sítě je nutné, aby základní prvky byly planární, tedy aby všechny body ležely v rovině, což není těžké u čtyř- a více-úhelníků porušit. Naopak u trojúhelníků je tato vlastnost zřejmě zaručena vždy.
- Obdobně je u trojúhelníků vždy zaručena konvexnost, zatímco u jiných základních prvků by bylo nutné vždy tuto vlastnost ošetřit.
- Každý konvexní mnohoúhelník lze rozdělit na trojúhelníky, přičemž je možné (ale ne nutně) získat lepší aproximaci původního objektu.
- Uniformita trojúhelníků, která umožňuje na trojúhelníkové sítě aplikovat různé operace se zárukou, že existují relativně snadné důkazy[11].



Obr. 9: Trojúhelníková a čtyřúhelníková síť'[12]
Obr. 10: Wireframe model[13]

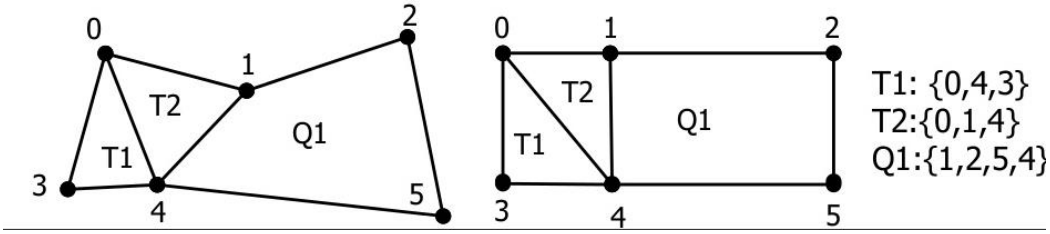
[11] HUGHES, John F. *Computer graphics: principles and practice*. Third (s.188)
[12] Dostupné z:
http://softimage.wiki.softimage.com/xsidocs/polys_QuadrangulatingandTriangulatingPolygons.htm
[13] Dostupné z: https://cdn.techterms.com/img/lg/wireframe_1280-2.jpg

3.1 Základní pojmy

Vlastnosti povrchových sítí[14]:

- Pevnost (solidity): Síť představuje pevný objekt v případě, pokud její plošky (faces) uzavírají kladný a konečný objem prostoru.
- Propojenost (Connectedness): Mřížka je propojená, pokud existuje nepřerušená trasa podél hran mezi jakýmikoliv dvěma body. Pokud mřížka není propojedná, zpravidla je uvažována jako dva objekty.
- Jednoduchost (Simplicity): Mřížka je jednoduchá, pokud neobsahuje díry.
- Rovinnost (Planarity): Mřížka je planární, pokud je každá ploška, která ji tvoří, planární. Rovinnost je požadována vždy.
- Konvexnost (Convexity): Mřížka je konvexní, pokud úsečka spojující jakékoliv dva body leží celá uvnitř objektu.

Jak bylo výše zmíněno základními prvky sítě jsou body propojené hranami do trojúhelníků. Pojďme si tedy představit ještě dva důležité pojmy: geometrie a topologie. *Geometrie* sítě představuje polohy jednotlivých bodů. *Topologie* odpovídá vztahům mezi body, hranami, ploškami. Rozdíl mezi topologií a geometrií můžeme vidět na (Obr.11), kdy obě sítě mají stejnou topologii ale různou geometrii[15].



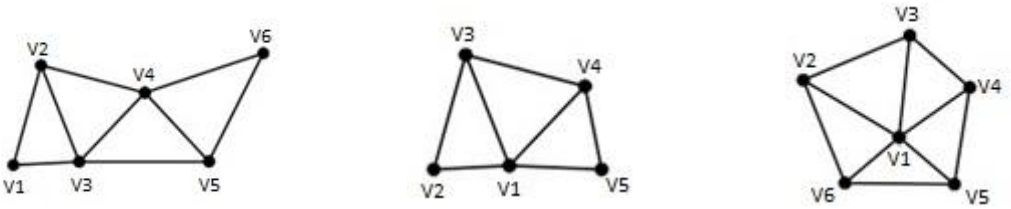
Obr. 11: Rozdíl mezi topologií a geometrií

Neméně důležitými pojmy v tvorbě povrchových sítí jsou základní podstruktury sítě:

- Pás (*Strip*) - Pokud trojúhelník f_1 je tvořen body v_1, v_2, v_3, pak trojúhelník f_2 je tvořen body v_2, v_3, v_4 (Obr.12 a).
- Vějíř (*Fan*) - Pokud trojúhelník f_1 je tvořen body v_1, v_2, v_3, pak trojúhelník f_2 je tvořen body v_1, v_3, v_4 (Obr.12 b).
- Hvězda (*Star*) - Speciální případ vějířové struktury, kdy f_n je tvořen body v_1, v_{n-1}, v_2 (Obr.12 c).

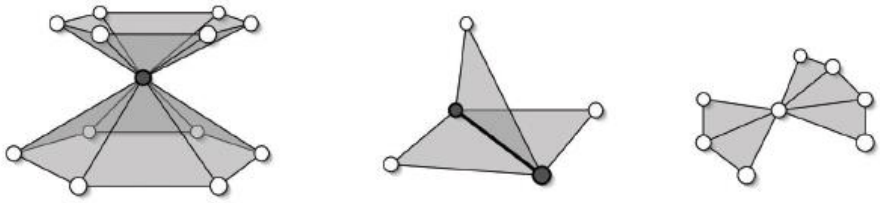
[14] *3D Object Representations* [online]. In: . 2012 [cit. 2018-05-25]. Dostupné z: http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/meshes/polygon_meshes.html

[15] PIROUZRIAN, Pirouz. Polygon Mesh Representation. In: *Slideshare* [online]. 2016 [cit. 2018-05-25]. Dostupné z: https://www.slideshare.net/NouPirouzrian/polygon-mesh-representation



Obr. 12: a) Pás b) Vějíř c) Hvězda[16]

Aby objekt mohl být reprezentován povrchovou sítí musí se jednat o *varietu* (manifold). Konečná rovinná povrchová síť je varietou, pokud hrany (e_i) a trojúhelníky (t_i) obsahující vrchol *v* mohou být seřazeny v cyklickém pořadí t_1, e_1,t_2, e_2,...t_n, e_n bez opakování tak, že hrana e_i je hranou mezi trojúhelníky t_i a t_{i+1}. Odtud plyne, že pro každou hranu existují právě dva trojúhelníky, které ji obsahují[17]. Snáz řečeno objekt je varietou, pokud neobsahuje žádné nevarietní hrany nebo body a zároveň sám sebe neprotíná. Nevarietní hrana je taková hrana, která obsahuje více jak 2 přidružené trojúhelníky. Nevarietní bod je incidentní více než jednomu vějíři trojúhelníků[18] (Obr.12). Jelikož nevarietní sítě jsou pro většinu algoritmů problematické budeme dále předpokládat, že reprezentované objekty jsou varietami.



Obr. 13: Nevarietní povrchové sítě[19]

Když mluvíme o povrchových sítích, je na místě zmínit Descartův-Eulerův vzorec pro mnohostěny (Descartes-Euler polyhedral formula) vyjadřující vztah mezi počtem bodů *V*, hran *H* a plošek*F* na uzavřené a propojené mřížce[20]:

$$V - E + F = 2(1 - g)$$

kde g je tzv. genus, což je číslo představující počet děr v tělese. Například koule tedy bude mít g=0, zatímco torus g=1. Speciálně pro trojúhelníkové mřížky lze odvodit další zajímavé vlastnosti[21]:

- Počet bodů je dvojnásobkem počtu trojúhelníků $F \approx 2V$
- Počet bodů je trojnásobkem počtu hran $E \approx 3V$
- Průměrná valence bodu (počet incidentních hran) ≈ 6

[16] PIROUZRIAN, Pirouz. Polygon Mesh Representation. In: *Slideshare* [online]. 2016 [cit. 2018-05-25]. Dostupné z: https://www.slideshare.net/NouPirouzrian/polygon-mesh-representation
[17]HUGHES, John F. Computer graphics: principles and practice. (193)
[18] BOTSCH, Mario. *Polygon mesh processing*. (s 12)
[19] BOTSCH, Mario. *Polygon mesh processing*..PMP(s 12)
[20]Weisstein, Eric W. "Polyhedral Formula." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/PolyhedralFormula.html
[21]BOTSCH, Mario. *Polygon mesh processing*.. (s 12)

3.2 Typy topologických struktur

U topologických struktur povrchových sítí jsou důležitými parametry spotřebovaný úložný prostor, čas potřebný na předzpracování struktury, k zodpovězení jakéhokoliv dotazu, k provedení konkrétního příkazu, náročnost na paměť a redundance (nadbytečnost) dat. Z tohoto důvodu jsou navrženy různé druhy datových struktur, které tyto parametry zohledňují. Než se zaměříme na konkrétní popisy jednotlivých struktur, pojďme si nejdříve představit základní operace nad sítí, které jsou součástí většiny algoritmů a které tedy při vyhodnocování kvalit jednotlivých topologií musíme brát v potaz. Jedná se o[22]:

- Přístup do všech bodů, hran a trojúhelníků včetně výčtu všech prvků ve stanoveném pořadí.
- Orientovaný průchod hranami trojúhelníku, což zahrnuje informace o předchozí a následující hraně trojúhelníku.
- Přístup k trojúhelníkům incidentním dané hraně.
- Přístup ke koncovým bodům dané hrany.
- Přístup z jakéhokoliv bodu, do incidentních hran nebo trojúhelníků.

Pomocí výše zmíněných operací (informací o vztazích mezi jednotlivými složkami) je možné lokálně i globálně procházet povrchovou sítí, sousedícími vrcholy, hranami a trojúhelníkovými plochami.

Informace o relacích (sousednosti) prvků sítě mohou být buď dodatečně odvozeny pomocí algoritmu (výpočtově náročnější, úspora úložného prostoru), nebo mohou být přímo uloženy v dané datové struktuře (menší nároky na výpočty, objemnější soubory). Nyní si představíme základní používané typy datových struktur využívaných pro práci s povrchovými sítěmi.

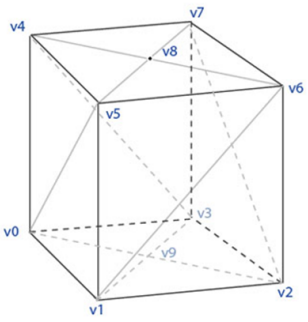
Topologie bod-bod (Vertex- Vertex topology)

Jedná se o nejjednodušší reprezentaci, kdy jsou každému bodu sítě přiřazeny jeho souřadnice v prostoru a indexy jeho sousedů (bodů, které s tímto bodem sdílí hranu) (Obr.14). Výhodou takovéto reprezentace je malá náročnost na úložný prostor a efektivní změna tvaru. Topologie bod-bod se však příliš nepoužívá, jelikož informace o hranách a trojúhelnících jsou dány implicitně, je třeba uložená data výpočetně náročně předzpracovat a vytvořit seznam trojúhelníků pro vykreslení.

[22] BOTSCH, Mario. *Polygon mesh processing* (23)

Vertex List

v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3



Obr. 14: Topologie bod-bod[23]

Topologie založené na trojúhelnících (Face based topology)

Dalším způsobem, jak uložit data o objektu jsou jednotlivé trojúhelníky reprezentované polohou jejich bodů.

- Pro jednodušší případy povrchových sítí je možné uložení pozic tří bodů do seznamu trojúhelníků. Vzhledem k absenci jakýchkoliv vazeb mezi trojúhelníky se takovéto struktuře říká též mnohoúhelníkový mix (v našem případě můžeme říci trojúhelníkový mix). Důsledkem nedefinovaných vztahů mezi trojúhelníky je mj. vysoká redundance (každý bod a jemu příslušná data jsou duplikována pro každý stupeň bodu[24]). Na tomto přístupu je založen například formát STL (zkratka slova stereolitography) mimo jiné využívaný v 3D tisku.
- Obdobnou datovou strukturou je indexovaná množina trojúhelníků (indexed face set), nebo též topologie sdílených vrcholů (shared vertex topology), která vysokou redundanci informací redukuje tak, že máme zvlášť seznam bodů daných svými souřadnicemi v prostoru a dále druhý seznam trojúhelníků obsahující pouze trojice indexů odkazující k jednotlivým bodům. Každý bod se tedy ve struktuře vyskytuje právě jednou. Díky své jednoduchosti a úspornosti místa je indexovaná množina trojúhelníků používána v řadě formátů souborů jako jsou OFF (object file format), OBJ (3D Wavefront object file) nebo VRML (Virtual Reality Modeling Language). Nevýhodou je stále náročné vyhledávání vztahů pro základní operace.

Triangles		
x_{11} y_{11} z_{11}	x_{12} y_{12} z_{12}	x_{13} y_{13} z_{13}
x_{21} y_{21} z_{21}	x_{22} y_{22} z_{22}	x_{23} y_{23} z_{23}
...	...	...
...	...	...
...	...	...
x_{F1} y_{F1} z_{F1}	x_{F2} y_{F2} z_{F2}	x_{F3} y_{F3} z_{F3}

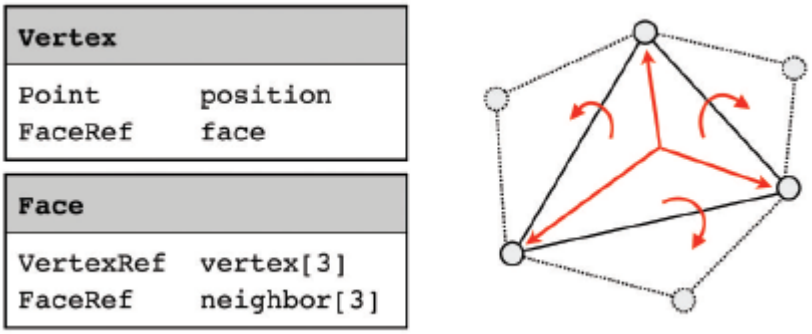
Vertices	Triangles
x_1 y_1 z_1	i_{11} i_{12} i_{13}
...	...
x_V y_V z_V	...
	...
	...
	i_{F1} i_{F2} i_{F3}

Obr. 15: Trojúhelníkový mix a indexovaná množina trojúhelníků

[23] Dostupné z: https://upload.wikimedia.org/wikipedia/commons/1/15/Vertex-Vertex_Meshes_%28VV%29.png

[24] BOTSCH, Mario. *Polygon mesh processing* (22)

- Standardní topologii založenou na trojúhelnících je taková, která má u každého trojúhelníku nejen indexy bodů, ale i sousedících trojúhelníků(Obr. 16). Dále každý bod má mimo své polohy v trojrozměrném prostoru uloženy i indexy všech trojúhelníků, které jej obsahují. Vidíme, že pomocí takto zadaných vztahů je již poměrně snazší provádět výše zmíněné základní operace. Standardní topologii využívá například známá C++ knihovna CGAL (computational geometry algorithms library). Můžeme si však všimnout, že ani v této struktuře nejsou zvlášť definovány hrany, není tedy možné na ně vázat data.

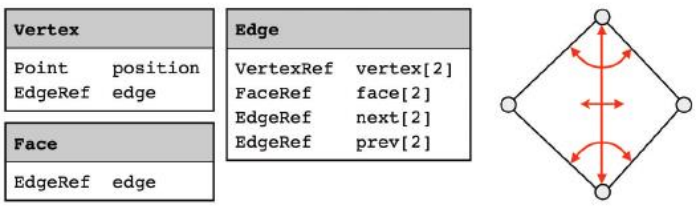


Obr. 16: Standardní datová struktura založená na trojúhelnících[25]

Topologie založené na hranách (Edge based topology)

Jak již název napovídá, hlavními prvky v těchto strukturách jsou hrany. Hranových struktur se využívá především tam, kde je za potřebí dynamický přístup k prvkům sítě (úpravy povrchové sítě).

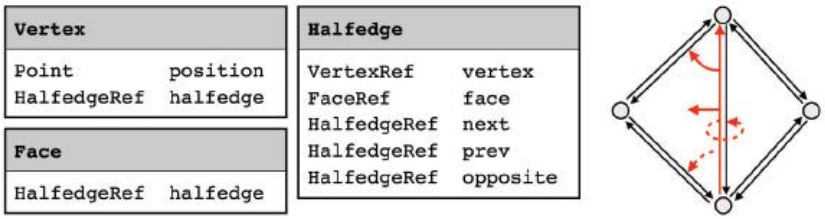
- Okřídlené hrany (Winged-edge topology) jsou struktury, obsahující indexované tabulky bodů, hran a trojúhelníků. Jádrovou strukturou jsou zde hrany, kdy každá hrana, která má v sobě uložena data o přilehlých trojúhelnících (odtud okřídlená hrana), okrajových bodech hrany a o navazujících hranách náležících přilehlým trojúhelníkům. Dále zde máme tabulky bodů a trojúhelníků, kdy každý bod a každý trojúhelník obsahuje seznam indexů přilehlých hran[26] (Obr.17). Nevýhodou jsou zřejmě větší nároky na paměť.



Obr. 17: Topologie okřídlených hran[27]

[25] BOTSCH, Mario. *Polygon mesh processing* (23)
[26] *3D Object Representations* [online]
[27] BOTSCH, Mario. *Polygon mesh processing* (23) (s24)

- Další možností, založenou na hranách je polohranová topologie (halfedge-based data structures). Všimněme si, že ve výše zmíněných topologiích není kladen důraz na pořadí, ve kterém jsou indexy ukládány do struktur bodů hran a trojúhelníků, což ztěžuje vyhledávání požadovaných sousedů. U polohranové topologie je proto každá neorientovaná hrana rozdělena na „poloviční“ hrany orientované proti směru hodinových ručiček okolo každého trojúhelníku[28]. Každá polohrana obsahuje informace na jaký bod míří, ke kterému patří trojúhelníku, předchozí a následující polohranu a svou opačnou (inverzní) polohranu (Obr.18).



Obr. 18: Polohranová topologie

[28] BOTSCH, Mario. *Polygon mesh processing*. (s25)

4 METODY ZJEDNODUŠOVÁNÍ POVRCHOVÉ SÍTĚ

V této kapitole si představíme základní metody zjednodušování povrchových sítí. Operace zjednodušení mají významné uplatnění v počítačové grafice, kdy chceme výrazně zmenšit počet trojúhelníků na povrchové síti. Například sejmeme-li pomocí 3D skeneru povrch trojrozměrného objektu, pak dostaneme bodový shluk o velkém množství bodů. Nad body pak provedeme triangulaci, čímž může vzniknout síť i o miliardách trojúhelníků. Nejenže jsou jakékoliv výpočty s takovou sítí náročné, ale v řadě případů jsou i zbytečné, jelikož síť zaznamenává detaily, které nejsou pro model nezbytné, případně je lidské oko ani nerozpozná. V zásadě jsou tři hlavní důvody, proč redukovat síť[29]:

- Eliminace redundance: Pokud v modelu existují velké plochy tvořené malými koplanárními trojúhelníky. Koplanární trojúhelníky je možné sdružit do mnohoúhelníku a ten pak opět triangulovat, což v závislosti na síti může mít za následek značné snížení počtu trojúhelníků na síti bez jakékoliv deformace.
- Úspora místa: Sítě o velkých počtech trojúhelníků jsou zpravidla náročné na místo na disku. Chceme-li síť zmenšit ve smyslu náročnosti na paměť, pak mimo nedestruktivního sdružování trojúhelníků můžeme volit i destruktivní algoritmy, tedy algoritmy, které síť aproximují s možností předem definované ztráty přesnosti.
- Zefektivnění vykreslování v reálném čase: Využívá se často ve vykreslování scén například v počítačových hrách, kdy u objektů, které jsou blíže, potřebujeme větší rozlišení, zatímco u objektů ve scéně vzdálenějších jsou třeba jen základní obrysy (LOD - level of detail).

Důvod, který nás vede k redukci sítě, by následně měl hrát roli ve zvoleném algoritmu.

Obecně můžeme zjednodušování povrchové sítě popsat jako operaci na mřížce (ne nutně trojúhelníkové), při které je původní mřížka $M=(V,F)$, kde V jsou vrcholy a F jsou trojúhelníky, nahrazena mřížkou M' geometricky nebo topologicky co nejvíce podobnou, avšak počet trojúhelníků je nižší[30]:

$$M'=(V',F'): \quad a)\ n=|V'|<|V| \ \&\ ||M-M'|| \text{ je minimální}$$
$$b)\ ||M-M'||<\varepsilon \ \&\ |V'| \text{ je minimální}$$

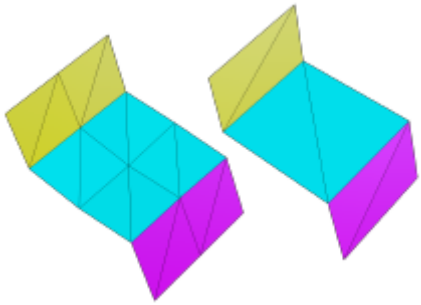
Zjednodušením dosáhneme reprezentace stejného povrchu s žádnými, nebo minimálními ztrátami[31]. Ztrátovost resp. aproximační chybaε je rozdíl mezi původní a zjednodušenou mřížkou. Aproximační chyba bývá stěžejní částí většiny sítě decimujících metod[32].Jednotlivé chybové metriky si představíme průběžně zároveň s algoritmy, ve kterých jsou využívány.

[29] LUEBKE, David P. A Developer's Survey of Polygonal Simplification Algorithms (s 24-35).
[30] ARVO, Jukka, *3D Mesh Simplification* (s.5)
[31]HUGHES, John F. Computer graphics: principles and practice (s 188)
[32] ARVO, Jukka, *3D Mesh Simplification* (s. 8)

4.1 Spojování koplanárních oblastí (Co-planar region merging)

Jak název napovídá, první algoritmus, který si představíme, spočívá ve zmenšování počtu trojúhelníků na síti pomocí spojování koplanárních nebo téměř koplanárních trojúhelníků do větších celků (Obr. 19).

Prvním krokem je nalezení téměř rovinných oblastí. Je vybrán trojúhelník a po jednom jsou testovány jeho přilehlé trojúhelníky (např. pomocí povoleného maximálního úhlu mezi trojúhelníky). Pokud je daný trojúhelník koplanární se svým sousedem, jsou tyto dva spojeny. Tak se pokračuje, dokud má vznikající mnohoúhelník neotestované sousedy. Následně je vzniklá planární oblast triangulována za vzniku stejného nebo menšího počtu trojúhelníků, než měla oblast původní (rovnost nastává např., je-li nalezená oblast čtyřúhelník, skládající se původně ze dvou trojúhelníků). Dále se vybere nový trojúhelník sítě a testují se jeho sousedé (přilehlé trojúhelníky). Aby se předešlo zacyklení algoritmu, dostanou trojúhelníky, jejichž všechny 3 okolní trojúhelníky byly otestovány na planaritu příznak a dále se neuvažují.



Obr. 19: Spojování koplanárních oblastí[33]

Velkou výhodou spojování koplanárních oblastí je, v závislosti na povolené odchylce, že se jedná o téměř bezztrátový algoritmus. Navíc je jeho princip velmi jednoduchý. Nevýhodou je, že ve větší míře redukuje počty trojúhelníků pouze v případě, že zadaný model obsahuje velké rovinné plochy (např. zdi budov) a je výpočetně náročný, tudíž se nehodí pro příliš komplexní sítě[34].

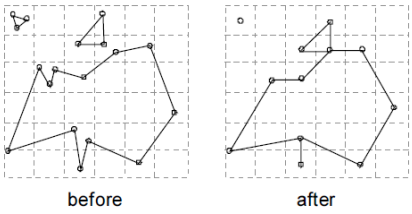
4.2 Bodové shluky (Vertex Clustering)

Bodové shlukování je algoritmus, který sdružuje blízké body povrchové sítě do tzv. shluků, které nahradí jediným, v rámci shluku nejvýznamnějším, bodem. Prvním krokem algoritmu je pokrytí trojrozměrného tělesa uniformní (nebo podle některých adaptací algoritmu neuniformní[35]) 3D mřížkou, jejíž hrany jsou rovnoběžné se souřadnými osami. Velikost jednotlivých dílků mřížky je zpravidla menší než

[33] ARVO, Jukka, *3D Mesh Simplification* (s10)
[34] ARVO, Jukka, *3D Mesh Simplification* (s11)
[35] ARVO, Jukka, *3D Mesh Simplification* (s12)

požadovaná maximální chyba ε. Dále jsou pro každou buňku nalezeny body, které do ní patří, a které jsou následně nahrazeny jediným bodem. Jak určit tento bod?

Možnosti určení nejvýznamnějšího bodu je více. Může se jednat o jeden z bodů originální sítě, zprůměrovaná hodnota původních bodů, nebo střed dané buňky[36]. Takto zvolené body však zřídka vedou ke kvalitním výsledkům. Lepší volbou proto bývá hledání optimální pozice reprezentanta pomocínejmenších čtverců, kdy předpokládáme, že pro dostatečně malou aproximační chybu ε je část sítě uvnitř buňky po částech plochá.Za po částech plochou část sítě považujeme takovou, kdyse normálykaždých dvou sousedících trojúhelníků (posunuty do společného vrcholu obou trojúhelníků) překrývají, nebo svírají malý úhel. Takto položené normály tvoří tzv. normálový kužel. V případě, že ε podsíť není po částech plochá, může být rozdělena na malý počet sektorů, které již po částech ploché jsou. Poloha reprezentanta je následně určena tak, aby jeho odchylka vůči všem (regresním) tečným rovinám, odpovídajícím sektorům podsítě, byla minimální. Pokud se všechny tečné roviny neprotínají v jednom bodě, pak je optimální pozice bodu hledána ve smyslu nejmenších čtverců[37].



Obr. 20: Shlukování bodů ve 2D[38]

Podstatnou nevýhodou bodových shluků je, že algoritmus nezachovává topologii. Dokonce se může stát, že takto zjednodušená síť již nebude varietou a to i v případě, že původní síť byla. To se může stát například když část povrchu, která je shluknuta do jediného bodu není homeomorfní vůči disku, resp. v případe, že dva různé úseky sítě prochází ε-buňkou[39]. Dále je nutno zdůraznit, že výsledek je nezanedbatelně závislý na posunutí ε-mřížky a nikdy není možno zjednodušit prvky větší než ε. Máme-li například plochu tvaru trojúhelníku rozdělenou na milion dílčích trojúhelníků, pak shlukováním bodů nikdy nedosáhneme redukce na jediný trojúhelník[40].

Metoda shlukování bodů je vhodná pro značně nadvzorkované 3D objekty, kdy není potřeba velkého stupně zjednodušení. Navíc je algoritmus velmi rychlý a použitelný obecně pro jakékoliv druhy sítí (i nevarietní)[41]. Ve speciálních případech, kdy je potřeba výrazně snížit komplexnost objektu, je možné za výhodu považovat i nezachování topologie. Například při snaze zachovat topologii povrchu houby, by

[36]Spojování koplanárních oblastí (12)
[37] BOTSCH, Mario. *Polygon mesh processing* (s. 114)
[38] FRANC, Martin. *Methods for polygonal mesh simplification*. (s 12)
[39] BOTSCH, Mario. *Polygon mesh processing* (s 113)
[40] FRANC, Martin. *Methods for polygonal mesh simplification*. (s 12)
[41] FRANC, Martin. *Methods for polygonal mesh simplification* (s 12)

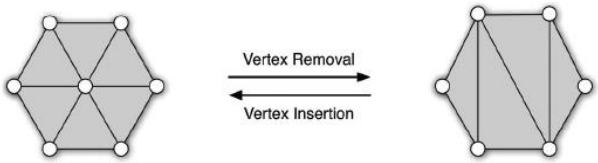
redukce jiným způsobem byla neefektivní, jelikož by musela zachovat veškeré povrchové díry, které houba obsahuje[42].

4.3 Inkrementální decimace (Incremental decimation)

Algoritmy využívající inkrementální destrukci sítě zpravidla cyklicky prochází komponenty sítě (body, hrany, trojúhelníky). Na základě zvoleného kritéria optimality je v každém cyklu zvolenakomponenta ke smazání. K mazání dochází dokud algoritmus nedosáhne ukončovacího kritéria, které často odpovídá míře zjednodušení sítě. Nyní si představíme operace s mřížkou, kterých se v inkrementální destrukci sítě využívá.

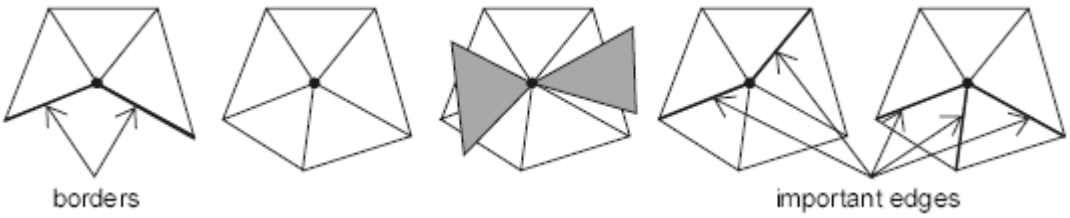
4.3.1 Odstraňování bodů (Vertex decimation)

Jak název napovídá, první operací je odstranění bodu povrchové sítě a následná triangulace vzniklé díry (Obr.21). Konkrétní algoritmy odstraňování vrcholů se v zásadě liší ve dvou aspektech. Jednak je to strategie, na základě které jsou voleny body k eliminaci (rovinnost, zakřivení, vzdálenost od původní povrchové sítě, nebo vzdálenost od průměru plochy, který je vypočítán na základě okolí bodu). Druhým je způsob retriangulace vzniklé díry. Ačkoliv řešení úlohy zaplnit vzniklou díru trojúhelníky má několik stupňů volnosti, výsledkem je vždy sít s o 1 méně vrcholy, o 3 méně hranami a o 2 méně trojúhelníky[43]. Retriangulace je triviální problém za předpokladu, že vzniklé díry jsou konvexní[44].



Obr. 21: Odstraňování bodů[45]

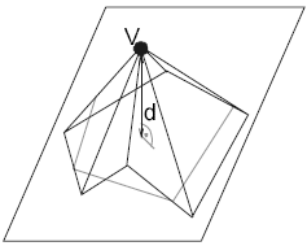
Přibližme si nyní, jakým způsobem je jednotlivým bodům přiřazována důležitost, resp. jak je stanoveno, zda je bod vhodný ke smazání. Důležitost bodu je určena především typem bodu. Pro obecnou povrchovou síť rozlišujeme 5 typů bodů (Obr. 22)



Obr. 22: Typy bodů zleva: hraniční, jednoduchý, komplexní, s vnitřními hranami a rohový[46]

[42] BOTSCH, Mario. *Polygon mesh processing* (s 113)
[43] BOTSCH, Mario. *Polygon mesh processing* (s 117)
[44] ARVO, Jukka, Antti. *3D MESH SIMPLIFICATION* (s 13)
[45] BOTSCH, Mario. *Polygon mesh processing* (117)
[46] FRANC, Martin. *Methods for polygonal mesh simplification* (s 14)

Jelikož v celém textu předpokládáme modely pevných (viz. základní vlastnosti sítí) trojrozměrných těles, pak si zřejmě z výše uvedených typů bodů vystačíme s tzv. jednoduchým bodem. Základním výpočtem důležitosti pro takový bod je vzdálenost od průměrové roviny(Obr. 23)



Obr. 23: Vzdálenost bodu od průměrové roviny[47]

Vzdálenost od průměrové roviny

Jedná se o vzdálenost, která je vypočtená z pozic sousedních bodů. Následovně[48]:

$$\vec{N} = \frac{\sum \vec{n}_i A_i}{\sum A_i}$$

$$\vec{n} = \frac{\vec{N}}{|\vec{N}|}$$

$$\vec{x} = \frac{\sum \vec{x_i} A_i}{A_i}$$

kde $\vec{n}_i$ jsou normály trojúhelníků incidentních danému bodu, x_i jejich těžiště a A_i obsahy. Pak:

$$d = |\boldsymbol{n}(\boldsymbol{v} - \boldsymbol{x})|$$

je vzdálenost bodu od průměrové roviny. Pokud se bod nachází do (uživatelem) určené vzdálenosti ε od průměrové roviny, pak může být smazán. Jak bylo výše zmíněno, body jsou promazávány cyklicky, přičemž se začíná od těch s nejmenší důležitostí, resp. vzdáleností *d*.

Hausdorffova vzdálenost

Dalším způsobem jak určit významnost bodu je Hausdorffova vzdálenost, která je přesnější, avšak výpočtově náročnější než měření vzdálenosti od průměrové roviny. Tato vzdálenost je určena maximální minimální vzdáleností dvou množin. Jinak řečeno máme-li dvě množiny $\boldsymbol{A}$ a $\boldsymbol{B}$, pak hledáme minimální vzdálenosti $d(a,\boldsymbol{B})$pro každé $a \epsilon \boldsymbol{A}$, ze kterých následně vybereme maximum[49]:

$$\mathrm{H}(\boldsymbol{A}, \boldsymbol{B}) = \max_{a \epsilon \boldsymbol{A}} \min_{b \epsilon \boldsymbol{B}} \|\mathrm{a} - \mathrm{b}\|.$$

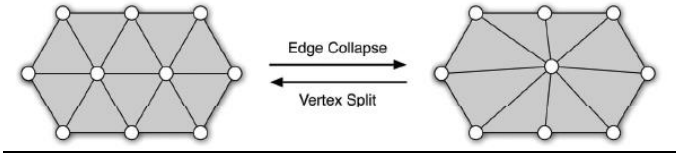
Můžeme si všimnout, že obecně se $\mathrm{H}(\boldsymbol{A}, \boldsymbol{B}) \neq \mathrm{H}(\boldsymbol{B}, \boldsymbol{A})$. Proto je rozlišována symetrická Hausdorffova míra, kdy je bráno maximum z těchto dvou hodnot, a jednostranná Hausdorffova míra, kdy je brána jen jedna z hodnot. V našem případě bývá využita jednostranná verze míry, kdy množinu $\boldsymbol{A}$ představují body původní sítě a množinu $\boldsymbol{B}$

[47] FRANC, Martin. *Methods for polygonal mesh simplification* (s 14)
[48] FRANC, Martin. *Methods for polygonal mesh simplification* (s 14)
[49] BOTSCH, Mario. *Polygon mesh processing* (s 121)

body sítě zjednodušené[50]. Je tedy nutné po celou dobu běhu algoritmu mít k dispozici originální síť.

4.3.2 Kolaps hran (Edge decimation/collapse)

Algoritmus je obdobou předchozího, avšak místo bodů jsou z povrchové sítě cyklicky odstraňovány hrany. Odstraňování se děje tak, že dva body, spojené hranou E{i,j}, jsou spojeny v jeden reprezentativní bod V{k} (Obr.24). Algoritmy založené na kontrakci hran se od sebe liší jednak v metodě pro určování hrany ke kolapsu a v určování polohy reprezentativního bodu. Tento bod může být určen například jako jeden z původních bodů hrany (někdy vyčleněn jako zvláštní kategorie[51]), střed hrany, libovolný bod ležící na původní hraně, nebo libovolný bod v okolí původní hrany[52].



Obr. 24: Kolaps hran[53]

Chybové kvadriky(Error Quadrics)

Mimo triviální způsoby, jak určit polohu reprezentanta byla navržena řada algoritmů lépe aproximujících původní síť. Nejpoužívanějším však jsou chybové kvadriky, které jsou kompromisem mezi přesností výstupní aproximace a rychlostí výpočtu.

V algoritmu je nejdříve stanovena pro každý bod původní sítě matice $\boldsymbol{Q}$ o rozměrech 4x4, určující vzdálenosti příslušného bodu od každého trojúhelníku na síti[54]:

$$\boldsymbol{Q} = \sum K_{\boldsymbol{p}}$$

$$\boldsymbol{K_p} = \boldsymbol{pp^t} = \begin{matrix} a^2 & ba & ca & da \\ ab & b^2 & cb & db \\ ac & bc & c^2 & dc \\ ad & bd & cd & d^2 \end{matrix}$$

kde $\boldsymbol{p} = [a, b, c, d]$, což jsou koeficienty rovnic rovin sousedících s bodem, pro nějž matici počítáme:

$$ax + by + cz + d = 0$$

a uvedeme do tvaru:

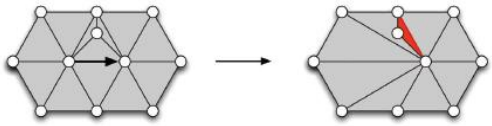
$$a^2 + b^2 + c^2 = 1.$$

[50]BOTSCH, Mario. *Polygon mesh processing* (121)
[51] BOTSCH, Mario. *Polygon mesh processing(121)*
[52] ARVO, Jukka, Antti. *3D MESH SIMPLIFICATION*
[53] BOTSCH, Mario. *Polygon mesh processing* (117)
[54] GARLAND, Michael a Paul S. HECKBERT. *Surface simplification using quadric error metrics*

Druhým krokem algoritmu je určení dvojic bodů $(\boldsymbol{v}_1, \boldsymbol{v}_2)$, které jsou uvažovány pro kontrakci. To jsou vždy body spojené hranou. Někdy se uvádí také dvojice bodů, pro které platí:

$$\|\boldsymbol{v}_1 - \boldsymbol{v}_2\| < \varepsilon$$

kde ε je pevně zvolená hodnota. Dále jsou vyloučeny takové dvojice bodů, které mají více než dva sousední body totožné, jelikož v takových případech může vzniknout nevarietní síť (Obr.25).



Obr. 25: Vznik nevarietní sítě[55]

Ve třetím kroku algoritmu je pro každý pár bodů vypočtena optimální poloha reprezentanta $\boldsymbol{v}'$, na základě minimalizace chybové metriky stanovené:

$$\Delta(\boldsymbol{v}) = \boldsymbol{v}^T \boldsymbol{Q} \boldsymbol{v}$$

Tato funkce je kvadratická a hledání minima je lineárním problémem, kdy řešíme soustavu rovnic:

$$\frac{\partial \Delta}{\partial x} = \frac{\partial \Delta}{\partial y} = \frac{\partial \Delta}{\partial z} = 0$$

což odpovídá[56]:

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{v}' = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

kde koeficienty q jsou brány z $\boldsymbol{Q}'$:

$$\boldsymbol{Q}' = \boldsymbol{Q}_1 + \boldsymbol{Q}_2 \,.$$

Potom $\boldsymbol{v}'$ je určen jako:

$$\boldsymbol{v}' = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

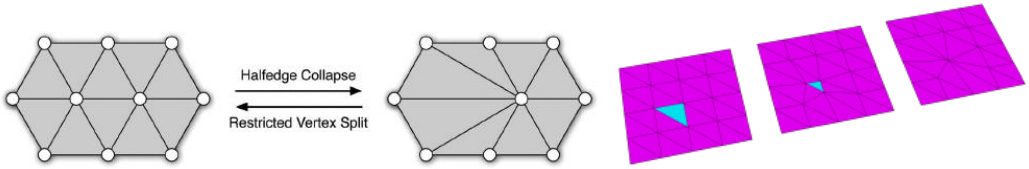
Může se však stát, že matice $\boldsymbol{Q}'$ nebude regulární a tedy nebude existovat její inverze. V takovém případě za bod $\boldsymbol{v}'$ volíme buď jeden z bodů $\boldsymbol{v}_1, \boldsymbol{v}_2$, nebo jejich průměr, v závislosti na hodnotě chybové metriky.

Následně jsou jednotlivé páry bodů seřazeny, podle velikosti chyby, která by byla způsobena kontrakcí $\Delta(\boldsymbol{v}')$ od nejmenší. Odtud se již jen iterativně odebírá po

[55] BOTSCH, Mario. *Polygon mesh processing* (s 119)
[56] GARLAND, Michael a Paul S. HECKBERT. *Surface simplification using quadric error metrics*

jednom pár, přičemž jsou aktualizovány vztahy nového uzlu$\boldsymbol{v'}$a uzlů sousedících (tedy jejich matice $\boldsymbol{Q}$, optimální poloha$\boldsymbol{v''}$ a chyba $\Delta(\boldsymbol{v''})$).

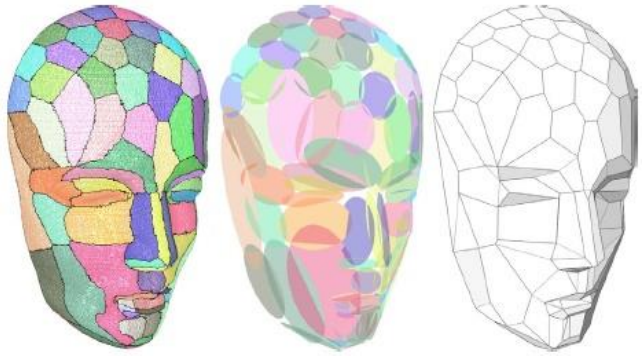
Pro úplnost zmiňme, že některé publikace (např.[57]) uvádí jako typ inkrementální decimace také poloviční kolaps hrany (Halfedge collapse) (Obr. 26), kdy je hrana "stažena" do jednoho z jejích krajních bodů. Jedná se tedy o speciální případ kolapsu hrany. Obdobně je speciálním případem kolapsu hrany i kolaps trojúhelníku ([58]) (Obr. 27)



Obr. 26: Poloviční kolaps hrany[59]
Obr. 27: Kolaps trojúhelníku[60]

4.4 Tvarová aproximace (Variational Shape Approximation)

Jedná se algoritmus, kdy je původní mřížka rozdělena na *k* oblastí R_i, které se vzájemně nepřekrývají. K daným regionům jsou nacházeny roviny, které tyto oblasti reprezentují (*proxies*) $P_i = (x_i, n_i)$, kde x_i je bod, který v dané rovině leží a n_i je jednotková normála roviny (Obr.28).



Obr. 28: Tvarová aproximace[61]

V algoritmu můžeme volit jednu ze dvou chybových metrik, které určují vzdálenost regionu R_i od jeho aproximace P_i. První z nich je založena na vzdálenosti bodu od roviny[62]:

[57]BOTSCH, Mario. *Polygon mesh processing*
[58] ARVO, Jukka, Antti. *3D MESH SIMPLIFICATION* (s. 16)
[59] BOTSCH, Mario. *Polygon mesh processing* (s. 117)
[60] ARVO, Jukka, Antti. *3D MESH SIMPLIFICATION* (s. 16)
[61] COHEN-STEINER. *Variational shape approximation* (s.1)
[62] COHEN-STEINER. *Variational shape approximation* (s. 3)

$$\mathcal{L}^2(R_i, P_i) = \iint_{r \epsilon R_i} \left\| r - \prod_i (r) \right\|^2 dr$$

kde $\prod_i(x)$ je ortogonální projekce bodu v argumentu funkce do roviny P_i,tedy:

$$\left\| r - \prod_i (r) \right\|^2 = min_{r \epsilon R_i, p \epsilon P_i} \|r - p\|^2 = \langle r - x_i, n_i \rangle^2$$

Druhou možnou metrikou je[63]:

$$\mathcal{L}^{2,1}(R_i, P_i) = \iint_{r \epsilon R_i} \|\boldsymbol{n}(r) - \boldsymbol{n}_i\|^2 \, dr$$

která je založena na měření normálového pole.

Cílem algoritmu je pak pro zadaný počet oblastí k globálně minimalizovat chybu:

$$E(R, P) = \sum_{i=1}^{k} E(R_i, P_i) \, .$$

Posledním krokem je pak extrakce zjednodušené sítě z aproximačních rovin pomocí triangulace.

Výše popsaný algoritmus zachovává důležité prvky sítě (rohy, ostré hrany, symetrii)[64]. Obecně se jedná o velmi dobrou aproximaci původního objektu, avšak za cenu ne příliš vysoké výpočetní efektivity[65].

[63] COHEN-STEINER,. *Variational shape approximation* (s. 4)
[64] BOTSCH, Mario. *Polygon mesh processing*. (s. 122)
[65] ARVO, Jukka, Antti. *3D MESH SIMPLIFICATION* (s. 6)

5 KNIHOVNY PRO ZJEDNODUŠOVÁNÍ POVRCHOVÝCH SÍTÍ

V této kapitole si stručně představíme nejvyužívanější softwarové knihovny, které zahrnují mimo jiných funkcí také zjednodušování trojúhelníkových povrchových sítí.

5.1 Open Mesh

Jedná se o knihovnu pro zobrazování a manipulaci s obecně mnohoúhelníkovými sítěmi. Je licencována (podle verze) pod LGPL v3 (Lesser General Public License) nebo BSD 3 clause licencí, v zásadě tedy volně použitelná, šiřitelná a modifikovatelná (za jistých podmínek).

Funkcemi knihovny je reprezentace, zjednodušování a vyhlazování povrchových sítí. Datová struktura je založena na polohranách, tedy orientovaných hranách, jejímiž atributy jsou bod, přilehlá plocha, následující hrana, protilehlá (opačně orientovaná) hrana(viz. kap. 3)[66].

Metodou pro zjednodušování povrchových sítí je zde poloviční kolaps hrany (v knihovně funkce *Decimater*), kdy je zvolen jeden z krajních bodů hrany, kam je celá hrana stažena. Dále zde existuje speciální atribut (*locked*), který zvolené body sítě vyjme ze zjednodušovacího procesu. Další zajímavou funkcí jenapříklad *is_manifold,* která zjistí, zda je zadaná síť varietou[67].

5.2 VTK

The visualization toolkit (VTK) je volně dostupný software pro zpracovávání 3D grafiky, zpracování obrazu a vizualizaci. Skládá se z knihovny napsané v jazyce C++ a rozhraní. Co se týče povrchových sítí, zjednodušování je podporováno pouze pro trojúhelníkové sítě, nicméně knihovna obsahuje také funkci pro triangulaci sítě (*vtkTriangleFilter*), která se neskládá výhradně z trojúhelníků. Z výše rozebíraných metod je možné ve VTK knihovně použít metodu *vtkDecimatePro*, což odpovídá kolapsu hran, přičemž chybovou metrikou je vzdálenost bodu a plochy. Druhou možností je *vtkQuadricDecimation,* což je opět kolaps hran, avšak chyba zjednodušené sítě je měřena pomocí chybových kvadrik[68].

5.3 CGAL

Computational Geometry Algorithms Library (CGAL) je velmi známou a obsáhlou C++ knihovnou, volně dostupnou pro nekomerční použití (LGPL/GPL licence). Knihovna

[66] Dostupné z: https://www.openmesh.org/intro/
[67] Dostupné z: http://www.openmesh.org/Documentation/OpenMesh-Doc-Latest/
[68] Dostupné z: https://www.kitware.com/products/books/VTKUsersGuide.pdf (122)

opět využívá metodu kolapsu hran, přičemž datovou strukturou je polohrana. Dále je implementovaný algoritmus určen výhradně pro trojúhelníkové sítě a je požadováno, aby síť byla varietou. Chybová funkce minimalizující soustavu až tří kvadratických rovnic. Každá z rovnic zohledňuje jeden z chybových aspektů: hranice (pokud síť hranici má), změna objemu mřížky, upřednostňování rovnostranných trojúhelníků před podlouhlými[69].

Dalšími knihovnami s implementovaným zjednodušováním povrchových sítí jsou:

- MeshDecimator
- GTS
- Libigl
- Trimesh

[69] Dostupné z: https://doc.cgal.org/latest/Surface_mesh_simplification/index.html

6 ŘEŠENÍ

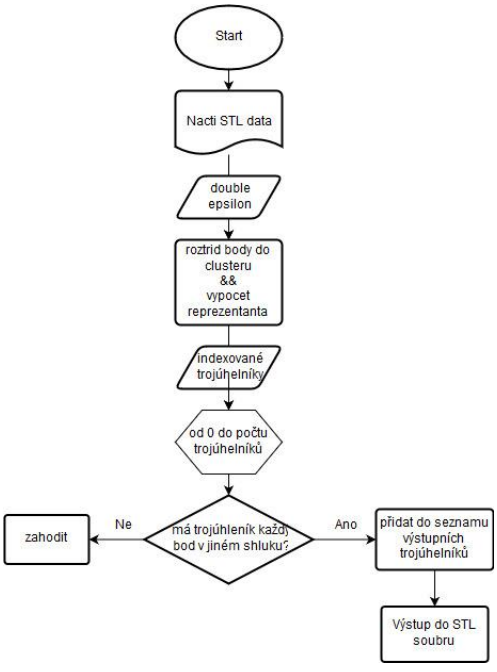
V rámci práce byly implementovány dva z výše teoreticky popsaných algoritmů. Prvním bude shlukování bodů, kdy za aproximaci bodů v daném shluku vezmeme průměrnou hodnotu bodů, které do něj náleží. Druhým vybraným algoritmem je kolaps hrany s měřením kvadratické chyby. Poznamenejme ještě že úvodní část obou z algoritmů - načtení bodů a trojúhelníků z STL souboru- nebude detailně principiálně popisována, jelikož byla poskytnuta firmou B&R.

6.1 Bodové shluky (Vertex Clustering)

Jak bylo již v teoretické části zmíněno, bodové shluky jsou asi nejsnadněji implementovatelným algoritmem s nejistými výsledky. Jeho jednoduchost můžeme vidět i na vývojovém diagramu.

Vstupem algoritmu je cesta k STL souboru, jehož počet trojúhelníků chceme redukovat, název výstupního souboru a ε - velikost jedné buňky. Vzorové volání funkce:

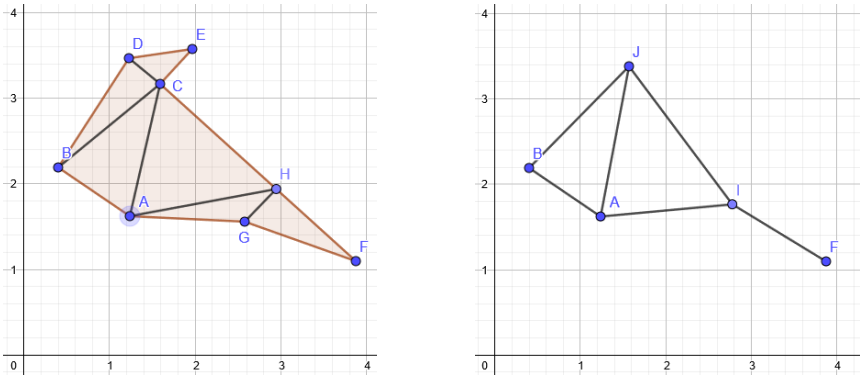
```
vertexClustering("C:\\Users\\User1\\Desktop\\teapot.stl", "object.stl", 1.0);
```



Pro algoritmus byla zvolena jakožto nejvýhodnější struktura indexovaná množina trojúhelníků. Budeme tedy mít dynamické pole bodů, a dynamické pole trojúhelníků. Pole bodů bude osahovat strukturu CVertex, která kromě souřadnic bodu obsahuje i trojici celých čísel, která bude značit, do jakého shluku příslušný bod patří. Pole trojúhelníků pak obsahuje trojice indexů odkazující na vrcholy trojúhelníku.

Ke konstrukci výše popsané struktury nejdříve načteme data z STL souboru. Data jsou reprezentována polem trojic hodnot typu double, které představují body modelu. Následně je zavedena funkce *getUniqVertAndTrianglesList()*. Vstupem funkce je zmiňované pole bodů a vstupně výstupními proměnnými je seznam unikátních bodů a pole indexovaných trojúhelníků. Pole unikátních bodů je získán za použití asociativního kontejneru map. V původním neuspořádaném poli bodů každá trojice bodů tvoří trojúhelník. Nahradíme-li jednotlivé body jejich indexy z pole unikátních bodů, dostáváme indexovanou množinu trojúhelníků.

Nad získanou datovou strukturou již můžeme aplikovat algoritmus. V naší implementaci jsme jej zavedli jako funkci *vertToClusters()*. Vstupem funkce jsou pole datové struktury a hodnota ε. Vstupně výstupními proměnnými je pole nedegenerovaných trojúhelníků a mapa shluků. Za nedegenerované trojúhelníky považujeme takové, jejichž každý bod leží v jiném shluku. V opačném případě totiž jednotlivé trojúhelníky mění v hrany, případně body. To můžeme vidět názorně ve dvou dimenzích na (Obr.29).



Obr. 29: Degenerace trojúhelníku CDE na bod a BCD a FGH na hrany

Ve funkci *vertToClusters()* procházíme postupně všechny body sítě a určujeme do jakého shluku patří. Shlukem rozumíme jednu buňku trojrozměrné sítě nad sítí. Zřejmě není třeba definovat každou takovou buňku v případě, že v ní neleží žádný bod jako například $< 0,1 > \times < 0,1 >$ v (Obr. 29). Z tohoto důvodu zakládáme jednotlivé buňky až ve chvíli, kdy jsou potřeba. Buňku reprezentuje trojice celých čísel, každé pro jednu osu. Mějme tedy bod o souřadnicích X,Y,Z. Pak shluk, ve kterém leží, je definován, jako:

$$s = [\text{floor}\left(\frac{X}{\varepsilon}\right), \text{floor}\left(\frac{Y}{\varepsilon}\right), \text{floor}\left(\frac{Z}{\varepsilon}\right)]$$

přičemž funkce *floor()* v c++ zaokrouhluje funkční hodnotu argumentu k nižšímu celému číslu. Pokud si tedy ještě jednou vezmeme (Obr.29), pak například bod A by patřil do shluku [1,1] zatímco body C,D i E do shluku [1,3]. Shluky *s* přitom ukládáme nejen do struktury CVertex, ale i do asociativní mapy *m*, abychom následně mohli dopočítat reprezentanta shluku, který poté nahradí všechny body v něm. Klíči mapy jsou tedy jednotlivé shluky a hodnotou uloženou na pozice klíčů je dvojice: souřadnice bodu a počet bodů. Pokaždé když do shluku ukládáme bod, tak jeho souřadnice přičítáme k souřadnicím předešlých uložených bodů a informaci o počtu bodů zvýšíme o 1. Odtud po dokončení iterace přes jednotlivé body již snadno získáme reprezentanty shluků.

Posledním krokem popisované funkce je iterace přes pole indexovaných trojúhelníků, kdy u každého trojúhelníku kontrolujeme, zda žádné dva jeho body neleží ve stejném shluku. To provádíme postupným porovnáním složek shluků *s*, uložených v příslušných bodech. Pokud trojúhelní nedegeneroval, tak jej zapíšeme do redukované indexované množiny trojúhelníků.

Nyní už nám stačí zapsat nalezenou redukovanou síť do souboru STL. K tomu slouží funkce *writeToSTL()*. Vstupem funkce je indexovaná množina redukovaných trojúhelníků, unikátní body, mapa shluků a výstupní soubor. STL soubory obecně mohou mít buď binární, nebo ASCII podobu. Náš výstupní soubor je ASCII, přičemž má následující strukturu. Začátek souboru je uveden slovem *solid* a konec je označen jako *endsolid*. Každý trojúhelník je definován svou jednotkovou normálou a třemi trojicemi souřadnic bodů (Obr. 30). Ve funkci tedy cyklicky procházíme redukované trojúhelníky. Pomocí indexů bodů zjistíme, do jakého shluku patří a z mapy shluků vypočteme konkrétní hodnotu každého bodu. Z těchto bodů vypočteme jednotkovou normálu a zapíšeme do souboru (Obr. 31)

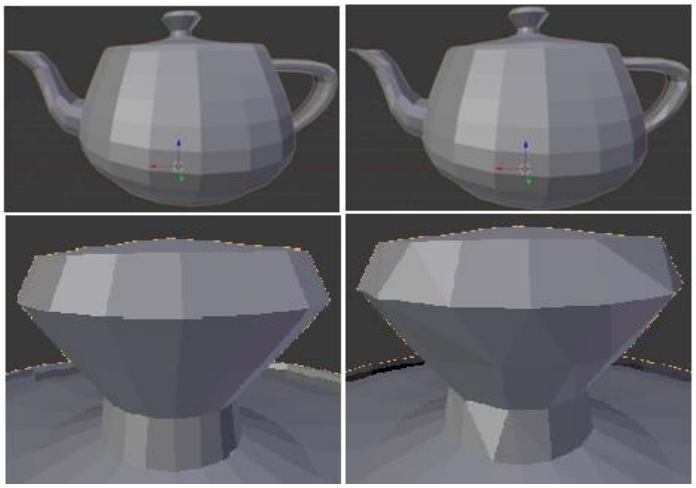
```
solid
   facet normal 0.000000e+000 0.000000e+000 1.000000e+000
      outer loop
         vertex 4.250000e+001 -8.036806e+001 1.070000e+002
         vertex 5.012963e+001 -9.838494e+001 1.070000e+002
         vertex 6.443289e+001 -6.443289e+001 1.070000e+002
      endloop
   endfacet
                              •
                              •
                              •
   facet normal 1.000000e+000 -1.156767e-016 0.000000e+000
      outer loop
         vertex -4.250000e+001 7.600000e+001 8.300000e+001
         vertex -4.250000e+001 8.036806e+001 1.070000e+002
         vertex -4.250000e+001 7.600000e+001 1.090403e+002
      endloop
   endfacet
endsolid
```

Obr. 30: Struktura ASCII STL souboru

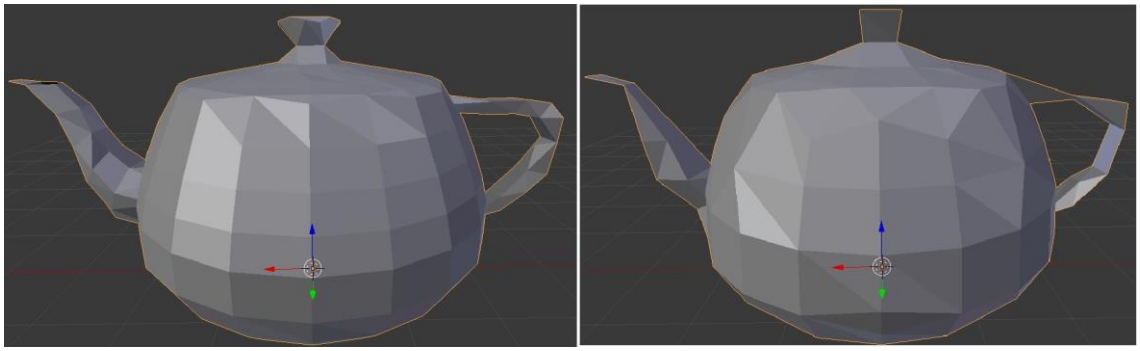
```
stlFile << "facet normal " << n[0] << " " << n[1] << " " << n[2] << '\n';
stlFile << "outer loop" << '\n';
stlFile << "vertex " << A.X << " " << A.Y << " " << A.Z << '\n';
stlFile << "vertex " << B.X << " " << B.Y << " " << B.Z << '\n';
stlFile << "vertex " << C.X << " " << C.Y << " " << C.Z << '\n';
stlFile << "endloop" << '\n';
stlFile << "endfacet" << '\n';
```

Obr. 31: Zápis bodů do souboru

K podrobnější analýze výsledků algoritmu se vrátíme v diskusní části tohoto textu. Nyní si ukažme alespoň ilustrační výsledek.Na (Obr. 32) vlevo vidíme originální model konvičky o 1560 trojúhelnících. Vpravo je pak tatáž konvička redukována s ε=0.1 na 1266 trojúhelníků. Na první pohled není vidět významnější rozdíl. Ten vidíme až při větším přiblížení na jemnější části modelu. V dalším obrázku je síť redukována s ε=0.3 na 697 trojúhelníků a ε=0.5 na 363 trojúhelníků. Zde již je redukce znát, přičemž vlevo na obrázku je patrné, že původně objemová síť místy “splaskla” na rovinnou. Na druhou stranu i při redukci na přibližně 23.26% původní sítě stále můžeme poznat, že se jedná o konvičku.



Obr. 32: Původní model 1560t a model s ε=0.1, 1266t



Obr. 33: ε=0.3, 679t a ε=0.5, 363t

6.2 Chybové kvadriky (Error quadrics)

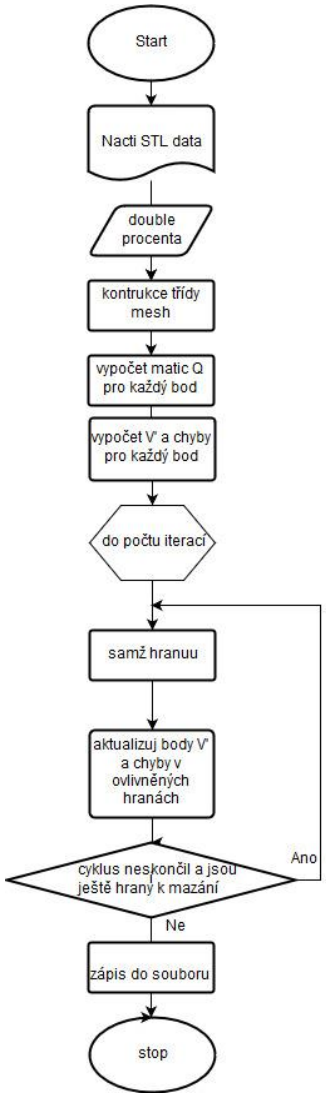
Druhým implementovaným algoritmem je decimace hran za použití chybových kvadrik. Tento algoritmus je podstatně implementačně složitější než bodové shluky, avšak zjednodušené sítě by měly být podstatně věrnějšími aproximacemi sítí původních. Funkci voláme následovně:

```
decimate("C:\\Users\\User1\\Desktop\\teapot.stl", "object.stl", 70.0);
```

Hodnotu ε z minulého algoritmu nahradila procenta, která značí, na kolik procent z původního počtu trojúhelníků chceme síť redukovat. Vývojový diagram vidíme na obrázku níže:

Začátek algoritmu je obdobný, jako u bodového shlukování, opět je proto naším výchozím bodem mapa indexů unikátních bodů a jejich souřadnic a asociované pole trojúhelníků. U bodových shluků nám taková struktura bohatě stačila, nicméně v tomto případě volímesofistikovanější strukturu. Protože redukujeme hrany, je vhodné mít z každé hrany přístup do jejích bodů a přilehlých trojúhelníků, které jsou likvidací hrany

zřejmě také ovlivněny. Je tedy logické založit hranovou strukturu.To se děje v konstruktoru třídy *mesh* postupným projitím všech trojúhelníků z asociovaného pole indexovaných trojúhelníků. Takto získáme čtyři důležité atributy sítě:



- mapa hran *EdgesWithInfo*:v ní jsou uloženy struktury *CEdge*. *CEdge* představují kompletní informace o hraně: indexy jejích bodů a přilehlých trojúhelníků. Struktura obsahuje i bod V'(Vc) do kterého může být daná hrana zborcena, a chybu *Error*, která by vzniklla při potenciálním kolapsu do Vc, tyto hodnoty jsou však dopočítány v algoritmu až později.
- mapa struktur *CTriangles*: trojúhelníky obsahují trojice indexů hran, kterými jsou tvořeny
- mapa *triAdjToPoint:* zde jsou pro každý index bodu sestaveny, asociativní pole indexů trojúhelníků, které daný bod obsahují.
- mapa *edgAdjToPoint*: je obdobou předchozí mapy, kdy bodům přiřazujeme seznamy hran, které je obsahují

Nyní se dostáváme k jádru algoritmu tak, jak je popsán v teoretické části. Nejdříve tedy voláme funkci *getVertexMatrices()*, která za pomoci vstupů (sítě a unikátních bodů) vytvoří každému bodu sítě matici Q. Ve funkci využíváme *triAdjToPoint*. Procházíme trojúhelník po trojúhelníku, přičemž z každého získáme jeho body a následně koeficienty roviny přilehlého trojúhelníku *a,b,c,d*. Z nich sestavujeme koeficienty dílčí matice K, které rovnou přičítáme ke koeficientům matice Q.

Druhým krokem algoritmu je výpočet bodu V' tak, aby minimalizoval chybu, které se dopustíme, pokud hranu tímto bodem nahradíme. V implementaci je tato funkce nazvána *getErrors()*. Nyní postupně procházíme hrany *EdgesWithInfo*. Na hranu nejdříve voláme funkci *flipCheck*(), která vrátí hodnotu *true* pokud decimací hrany vznikne nevarietní síť. Tehdy nemá smysl příslušné hodnoty počítat, protože ke kolapsu hrany nedojde. Samozřejmě s různými úpravami sítě se může tato vlastnost změnit a bod V' i chyba jsou dodatečně dopočítány. Pokud hrana projde kontrolou, pak pro ni vypočítáme matici Q' (Qc), která odpovídá součtu matic Q krajních bodů hrany. Současně vytváříme matici Qcd, což je matice Qc s posledním řádkem [0 0 0 1]. Pokud existuje inverzní matice Qcdi k matici Qc, pak ji nalezneme pomocí funkce *inverse()*[70] a vypočítáme polohu Vc. Jak je zmíněno v teorii, pokud inverzní matice neexistuje, volíme jako bod Vc jeden z krajních bodů hrany nebo jejich průměr podle velikosti chyby. Každou hranu, která projde funkcí *flipCheck()*navíc zařadíme do asociovaného pole VP dvojic (chyba, hrana). Po

[70]http://www.euclideanspace.com/maths/algebra/matrix/functions/inverse/fourD/index.htm

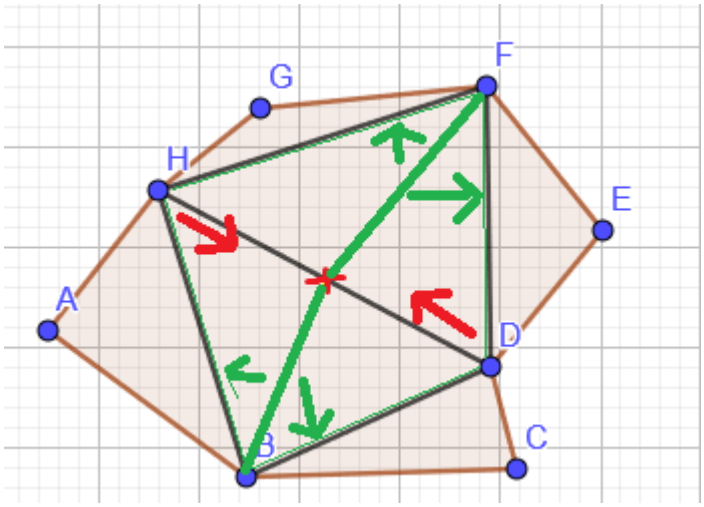
projití všech hran v mapě *EdgesWithInfo* pole VP seřadíme vzestupně podle chyby, čímž na první pozici pole získáme hranu nejvhodnější k decimaci.

Dostáváme se k nejdůležitější části algoritmu - funkci *deci()*. V ní nejdříve určíme počet iterací tak, aby výsledný počet trojúhelníků odpovídal požadovaným procentům:

$$počet\ iterací\ =\ round(\frac{\frac{počet\ trojúhelníků}{100}\ *\ (100\ -\ požadovaná\ procenta)}{2})$$

Výsledek dělíme dvěma, protože v každé iteraci odebíráme dva trojúhelníky. Z předchozí funkce máme již určenou hranu pro likvidaci. Kolaps provádíme následně:

- V mapě zakládáme nový bod, který odpovídá bodu Vc decimované hrany (červený bod).
- Určíme prvky sítě, které budeme mazat: hranu určenou ke kolapsu, její krajní body, přilehlé trojúhelníky a zbylé hrany přilehlých trojúhelníků (jelikož taková dvojice vždy splyne v jednu novou hranu).
- Vytvoříme nové hrany sjednocením nedecimovaných hran přilehlých trojúhelníků (v obrázku vyznačeny zeleně). Toho docílíme funkcí *createEdge()* použitou na každý přilehlý trojúhelník zvlášť. Funkce zároveň obstarává nahrazení hran společných s likvidovanými trojúhelníky v sousedních trojúhelnících hranami novými.

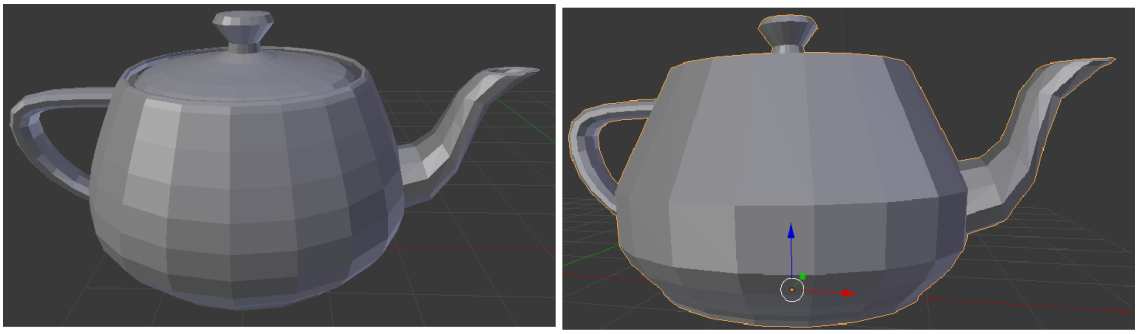


Obr. 34: Schéma tvorby nových hran

- Ve struktuře *edgAdjToPoint*, v bodech decimované hrany smažeme všech 5 zanikajících hran (tj. decimovanou hranu + pro každý trojúhelník 2 hrany, které jsou sjednoceny do nové) a vložíme nově vzniklé hrany.
- V okolních hranách mazané hrany nahrazujeme body mazané hrany bodem Vc.
- V *edgAdjToPoint* zavádíme nový bod Vc a vkládáme do něj jeho přilehlé hrany (tj. hrany které původně patřily ke krajním bodům hrany k odstranění).
- Mažeme body, hrany, trojúhelníky určené ke smazání.

Na závěr každého cyklu je pak zavolána funkce *updateEdges()*, která vypočítá/přepočítá bod Vc a chybu pro každou hranu přilehlou novému bodu. Zároveň aktualizuje a seřadí pole VP. Po skončení *deci()* proběhne funkcí *writeToSTL()* zápis do STL souboru stejně jako u algoritmu shlukování bodů.

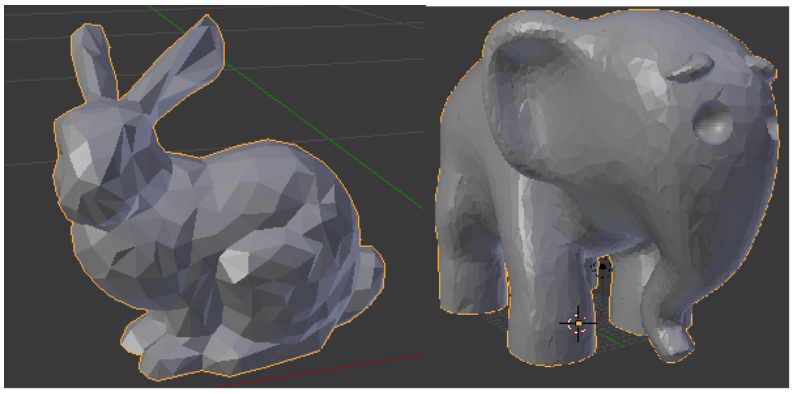
Podívejme se tedy na výstup algoritmu:



Na obrázku vidíme snížení počtu trojúhelníků algoritmem decimace hran na 80% původní hodnoty, tedy přibližně na 1249 trojúhelníků z původních 1560. Z obrázků vidíme, že si tento algoritmus s aproximací vede poměrně lépe, než algoritmus shlukování bodů. Avšak je třeba zmínit, že modle konvičky se celkově skládá ze čtyř s sebou vzájemně nepropojených částí: hlavní část, nálevka, ouško a pokličkа. S tím si tento algoritmus neporadí a redukuje pouze jednu z částí.

6.3 Aplikace implementovaných algoritmů

Algoritmy porovnáme na dvou různých sítích - bunny.stl (1000 trojúhelníků) a elephant.stl (14416 trojúhelníků) (Obr. 35)- pokaždé pro tři různé míry zjednodušení. Porovnávat budeme jednak podle časové náročnosti výpočtů a dále si pro každé zjednodušení vykreslíme obrázek a budeme srovnávat vizuální stránku zjednodušení. Hodnoty procentuálního zjednodušení a hodnoty ε budou voleny tak, aby příslušná zjednodušení generovala přibližně stejný počet trojúhelníků.



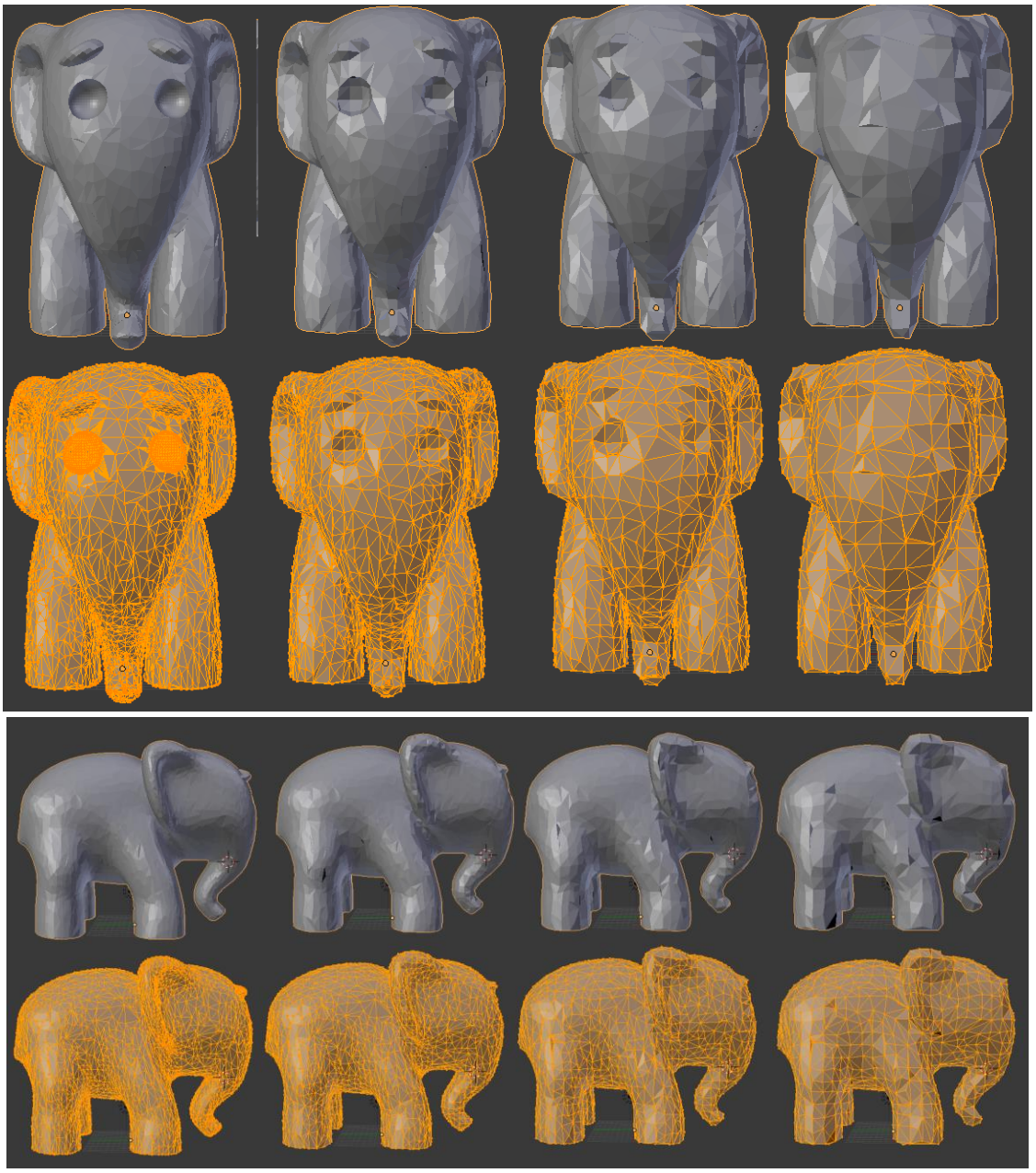
Obr. 35: Originální modely

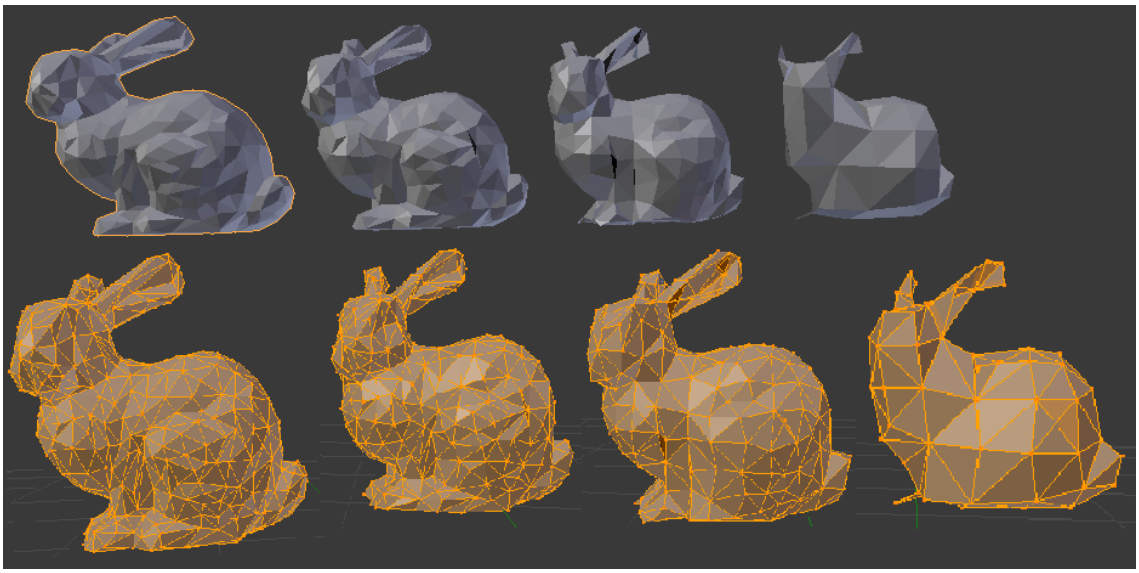
Tabulka výpočtové náročnosti:

	ε =1.0	**p =49.8**	**ε =1.5**	**p =30.9**	**ε =2.0**	**p =19.8**
Slon	272.6 ms	3 970 ms	205.0 ms	4 932 ms	122.4 ms	5 430 ms
	ε =0.02	**p =83.4**	**ε =0.03**	**p =55.2**	**ε =0.06**	**p =16.6**
Králík	46.1 ms	92.9 ms	30.0 ms	111.4 ms	31.6 ms	132.7 ms

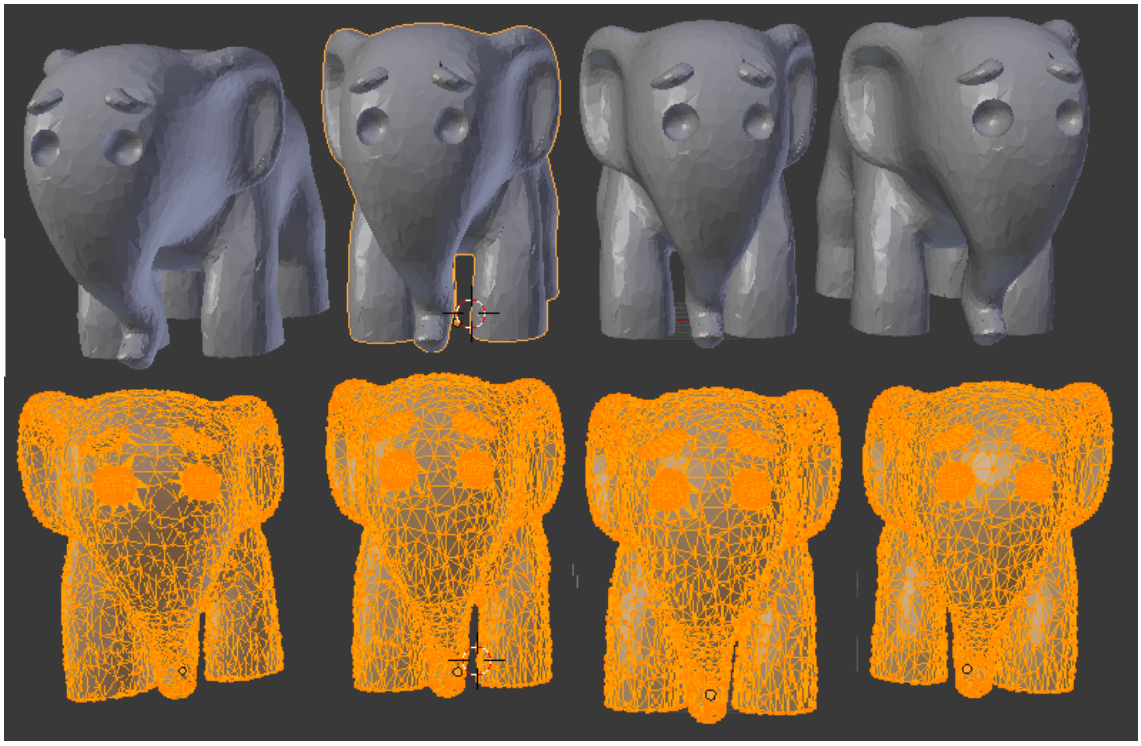
Grafický výstup:

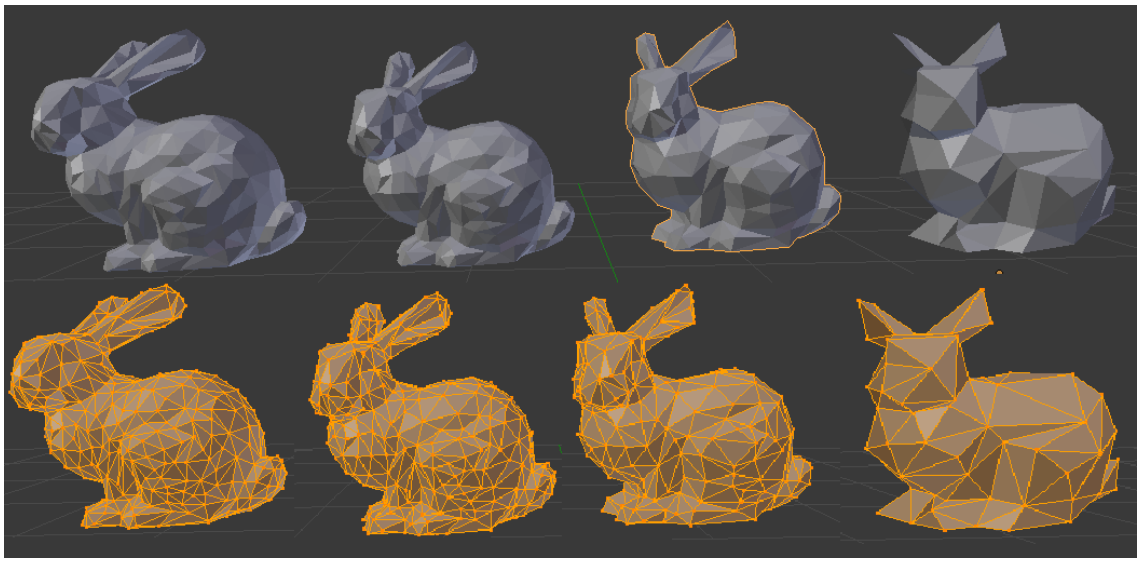
a) Bodové shluky





b) Decimace hrany





7 ZHODNOCENÍ A DISKUZE

Vidíme, že v případě kolapsu hrany u slona algoritmus negeneruje očekávané výsledky, resp. síť není redukována. Ačkoliv je síť na první pohled uzavřená (pro každou hranu existují právě dva přilehlé trojúhelníky), opak je pravdou a existující okrajové hrany sítě způsobily nefunkčnost celého algoritmu.V případě kolapsu hrany musíme brát velké ohledy na povahu sítě. To je velká nevýhoda například oproti shlukování bodů, které můžeme použít prakticky na jakoukoliv trojúhelníkovou síť.

Dále jsme se při měření doby běhu algoritmů a při pozorování změn na sítích ověřili to, co jsme se dozvěděli již v teoretické části. Tedy podíváme-li se na doby běhů,vidíme, že zvýšením počtu trojúhelníků sítě o asi 13 500 se doba běhu u shlukování bodů přibližně zešestinásobila (v řádu stovek milisekund). U kolapsu hrany je přibližně čtyřicetinásobná až do řádu tisíců milisekund. Tento úkaz můžeme jednoznačně přisoudit tomu, že zatímco bodové shlukování je algoritmus jednokrokový (jen jedinkrát za běh jsou vypočítány shluky, na základě kterých jsou pak opět právě jednou vypočítány redukované trojúhelníky). Na druhou stranu kolaps hrany je algoritmem rekurentním, a tedy se náročnost na výpočet s každým přibývajícím bodem nelineárně zvyšuje. Na výkon může mít vliv také např. vytíženost počítače, na kterém jsou algoritmy testovány, nicméně takto vzniklé odchylky jsou zanedbatelné.

Podíváme-li se všakobecně na grafické výsledky algoritmů, tak vidíme, že při redukci na stejný počet trojúhelníků vyváří algoritmus kolapsu hran důvěrnější aproximace sítě vůči původnímu tvaru. Velkou výhodou algoritmu decimace hran je navíc zadaná redukce sítě v procentech, respektive počtu trojúhelníků, které chceme odstranit. U ε velikosti shluku můžeme jen odhadovat, kolik trojúhelníků zmizí. Máme-li například malý model a zvolíme příliš velké ε, pak se nám může stát, že bude zredukován absolutně. Na druhou stranu zvolíme-li příliš malé ε, může být redukce nedostatečná, nebo žádná. Dále hraje roli,zda jsou trojúhelníky přibližně konstantní velikosti, nebo se síť skládá střídavě ze shluků malých trojúhelníků a velkých trojúhelníkových ploch.

Takto odlišné algoritmy se na druhou stranu mohou doplňovat například ve zmíněné síti s výrazně rozdílnými velikostmi trojúhelníků, kdy se pomocí bodových shluků odstraní malé trojúhelníky, které by při kolapsu hrany zatěžovaly paměť neúměrně vzhledem k výslednému vlivu na podobu sítě.

.

8 ZÁVĚR

V teoretické části práce jsme se nejdůsledněji věnovali algoritmům určeným pro redukci počtu trojúhelníků na povrchové síti, které jsme zavedli v kontextu počítačového modelování, pojmů z obecných povrchových sítí a existujících řešení. Praktická část se věnovala implementacím vybraných algoritmů, jejich aplikaci a dále diskuzí nad jejich použitelností a výsledky.

V první kapitole jsme rozebírali možnosti reprezentací objektů v počítači. Jednalo se o surová data získaná určitým způsobem měření, která se používají pro další zpracování. Dále jsme si zběžně zmínili reprezentace povrchů objektů pomocí povrchových sítí nebo povrchových křivek a objemové reprezentace jako voxely nebo konstruktivní geometrii pevných objektů (CSG).

Druhá kapitola je věnována již zcela povrchovým sítím. Nejdříve hovoříme obecně o mnohoúhelníkových sítích a jejich výhodách a nevýhodách oproti trojúhelníkovým sítím. Postupně se pak dostáváme k vlastnostem, strukturám a především topologiím výhradně trojúhelníkových sítí.

Ve třetí kapitole se zabýváme algoritmy pro redukci povrchových sítí a metrikami, které s algoritmy úzce souvisí. Začínáme topologii zachovávajícím algoritmem slučování koplanárních oblastí. Následuje shlukování bodů. Největší část je věnována cyklickým algoritmům, které v každé iteraci redukují v síti určitý počet trojúhelníků až po dosažení maximální chyby. Posledním zmiňovaným algoritmem je tvarová aproximace a její potenciální metriky.

Praktická část byla věnována implementovaným algoritmům bodových shluků a kolapsu hrna, dále ukázkou jejich aplikace a diskuzí nad získanými výsledky.

Hlavním přínosem práce jsou implementované algoritmy pro zjednodušování sítě ve formátu STL. Veškeré programy byly vytvořeny v jazyce C++ ve vývojovém prostředí Visual Studio 2015. Programy jsou přiloženy na datovém nosiči.

9 SEZNAM POUŽITÉ LITERATURY

LITERATURA

[1] HUGHES, John F. Computer graphics: principles and practice. Third edition. Upper Saddle River, New Jersey: Addison-Wesley, 2014. ISBN 978-0321399526.

[2] BOTSCH, Mario. *Polygon mesh processing*. Natick, Mass.: A K Peters, c2010. ISBN 15-688-1426-7.

[3] LUEBKE, David P. A Developer's Survey of Polygonal Simplification Algorithms. *IEEE Computer Graphics and Applications*. Dostupné z: https://www.cs.virginia.edu/~luebke/publications/pdf/cg+a.2001.pdf

[4] ARVO, Jukka, Antti EURANTO, Lauri JÄRVENPÄÄ, Teijo LEHTONEN a Timo KNUUTILA. *3D MESH SIMPLIFICATION: A survey of algorithms and CAD model simplification tests*. 2015. University of Turku Technical Reports

[5] FRANC, Martin. *Methods for polygonal mesh simplification*. Pilsen, 2002. University of West Bohemia.

[6] GARLAND, Michael a Paul S. HECKBERT. *Surface simplification using quadric error metrics* [online]. 1997 [cit. 2018-05-25]. DOI: 10.1145/258734.258849. ISBN 10.1145/258734.258849. Dostupné z: http://portal.acm.org/citation.cfm?doid=258734.258849

[7] COHEN-STEINER, David, Pierre ALLIEZ a Mathieu DESBRUN. *Variational shape approximation* [online]. 2004 [cit. 2018-05-25]. DOI: 10.1145/1186562.1015817. ISBN 10.1145/1186562.1015817. Dostupné z: http://portal.acm.org/citation.cfm?doid=1186562.1015817

WEBOVÉ ZDROJE

[1] In: *3Deling* [online]. [cit. 2018-05-25]. Dostupné z: http://www.3deling.com/whta-is-a-point-cloud/

[2] PolySoup surface node. In: *Sidefix* [online]. [cit. 2018-05-25]. Dostupné z: http://www.sidefx.com/docs/houdini12.5/nodes/sop/polysoup

[3] *3D Object Representations* [online]. In: . 2012 [cit. 2018-05-25]. Dostupné z: http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/meshes/polygon_meshes.html

[4] Solid modelling cg. In: *Slideshare* [online]. 2016 [cit. 2018-05-25]. Dostupné z: https://www.slideshare.net/Nareek/solid-modelling-cg

[5] PIROUZRIAN, Pirouz. Polygon Mesh Representation. In: *Slideshare* [online]. 2016 [cit. 2018-05-25]. Dostupné z: https://www.slideshare.net/NouPirouzrian/polygon-mesh-representation

[6] Dostupné z: https://www.openmesh.org/intro/

[7] http://www.openmesh.org/Documentation/OpenMesh-Doc-Latest/

[8] https://www.kitware.com/products/books/VTKUsersGuide.pdf

[9] Dostupné z: https://doc.cgal.org/latest/Surface_mesh_simplification/index.html

[10] http://www.euclideanspace.com/maths/algebra/matrix/functions/inverse/fourD/index.htm

ZDROJE OBRÁZKŮ:

[1] Bodové shluky: Dostupné z: https://www.codeproject.com/Articles/839389/Fast-Point-Cloud-Viewer-with-Csharp-and-OpenGL

[2] Hloubková mapa: Dostupné z: http://www.idl.rie.shizuoka.ac.jp/study/project/tof/index_e.html

[3] Trojúhelníková povrchová síť: Dostupné z: http://www.lix.polytechnique.fr/~maks/Verona_MPAM/TD/TD2/images/snapshot00.png

[4] NURBS reprezentace: Dostupné z: http://softimage.wiki.softimage.com/xsidocs/surfs_About_Surfaces.htm

[5] Princip konstruktivní objemové geometrie: http://groups.csail.mit.edu/graphics/classes/6.837/F98/talecture/

[6] Tvorba BSP stromu: http://www.connellybarnes.com/work/class/2015/intro_gfx/lectures/17-3DObjectRepresentation.pdf

[7] Trojúhelníková a čtyřúhelníková síť: http://softimage.wiki.softimage.com/xsidocs/polys_QuadrangulatingandTriangulatingPolygons.htm

[8] Wireframe model: https://cdn.techterms.com/img/lg/wireframe_1280-2.jpg

[9] Trojúhelníkový mix a indexovaná množina trojúhelníků https://upload.wikimedia.org/wikipedia/commons/1/15/Vertex-Vertex_Meshes_%28VV%29.png

[10] Standardní datová struktura založená na trojúhelnících: BOTSCH, Mario. *Polygon mesh processing* (s. 23)

[11] Topologie okřídlených hran: BOTSCH, Mario. *Polygon mesh processing* (s.23)

[12] Spojování koplanárních oblastí: ARVO, Jukka, *3D Mesh Simplification* (s11)

[13] Odstraňování bodů: BOTSCH, Mario. *Polygon mesh processing* (117)

[14] Vzdálenost bodu od průměrové roviny: FRANC, Martin. *Methods for polygonal mesh simplification* (s 14)

[15] Kolaps hran: BOTSCH, Mario. *Polygon mesh processing* (117)

[16] Vznik nevarietní sítě: BOTSCH, Mario. *Polygon mesh processing* (s 119)

[17] Poloviční kolaps hrany: BOTSCH, Mario. *Polygon mesh processing*

[18] Kolaps trojúhelníku: ARVO, Jukka, Antti. *3D MESH SIMPLIFICATION*

[19] Tvarová aproximace: *Variational shape approximation*

10 SEZNAM PŘÍLOH

CD